REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	October 28, 1998	Final Report 4/2/98 - 10/2/98

4. TITLE AND SUBTITLE

Portable Reusable Application Software
SBIR Phase I Final Technical Report

5. FUNDING NUMBERS

C N68335-98-C-0140
Item No. 0001AF

6. AUTHOR(S)

N. Carl Ecklund, Technical Director, MCCI

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Management Communications and Control, Inc. (MCCI)
2000 North 14th Street
Suite 220
Arlington, VA 22201

8. PERFORMING ORGANIZATION REPORT NUMBER

MCCI-98-NAWC-002

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Charles Bohman, Government Technical Liaison
Naval Air Systems Command HQ - Code PMA-299
Bldg. 2272, Suite 156
47123 Buse Rd., Unit IPT
Patuxent River, MD 20670-1457

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

19981110 015

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 Words)***

The need for application software portability and reusability has been increased by the COTS revolution. Operating system and math library independence are essential to portability strategies. However, in order to achieve the high throughput required by real-time sensor processing systems, the executable must be optimized for the specific target.

Management Communications and Control, Inc. (MCCI) has developed a methodology and a toolset which provides translation of target independent applications to target specific source code incorporating target optimized libraries. Application portability and reusability is inherent in the methodology. An order of magnitude reduction in application development time has been demonstrated. Life cycle costs should be reduced by at least the same factor. The methodology supports low cost reuse of the AN/UYS-2 code base. This report provides an overview of the methodology and the toolset. Porting of the DICASS sonobuoy signal processing from an AN/UYS-2 implementation to an implementation using the MCCI methodology and toolset is demonstrated.

14. SUBJECT TERMS

Autocoding Toolset, AN/UYS-2 Code Reuse, Open API, Sonar Signal Processing, Portable Software, Life Cycle Cost Reduction, Processing Graph Method (PGM), COTS

15. NUMBER OF PAGES

209

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

Portable Reusable Application Software for COTS Platforms
SBIR N98-030 Phase I
Program Progress Report - 0001AF

Contractor:
Management Communications and Control, Inc.
(MCCI)
2000 North 14th Street
Suite 220
Arlington, VA 22201

Contract Number: N68335-98-C-0140

Key Person: Christopher B. Robbins, President

Government Technical Liaison:
Naval Air Systems Command HQ
Attn: Mr. Charles Bohman - Code PMA-299
Bldg. 2272, Suite 156, 47123 Buse Rd., Unit IPT
Patuxent River, MD 20670-1457

Report Number: MCCI-98-NAWC-002

Portable Reusable Application Software SBIR Phase I Final Technical Report

October 28, 1998

Prepared for:
Naval Air Systems Command
Code PMA-299
Patuxent River, MD 20670-1457

Prepared by:
Management Communications and Control Inc. (MCCI)
2000 North Fourteenth Street, Suite 220
Arlington, VA 22201

Under Contract Number:
C68335-98-C-0140

Portable Reusable Application Software
SBIR Phase I Final Technical Report
Table of Contents

List of Figures

Acronyms

AG - Application Generator
API - Application Program Interface
CGA - Channel Gain Adjust
CP - Command Program
CP GUI - Command Program Graphical User Interface
CPI - Command Program Interface
DPAG - Domain Primitive Application Graph
EAG - Equivalent Application Graph
GIP - Graph Instantiation Parameter
GrM - Graph Manager
GrTT - Graph Translation Tool
GSMP - Graph execution Simulation on Multiple Processors
GV - Graph Variable
IOP - Input/Output Procedure
MPID - Multiprocessor Primitive Interface Description
MPIDGen - MPID Generator
NEP - Node Execution Parameter
PB - Partition Builder
PGM - Processing Graph Method
PID - Primitive Interface Description
PIP - Primitive Interface Procedure
PLU - Primitive Library Unit
SAR - Synthetic Aperture Radar
SPGN - Signal Processing Graph Notation
SRTS - Static Run-Time System
TPM - Target Primitive Map

Portable Reusable Application Software

1. Introduction

As the U. S. Navy transitions to COTS based systems, the need for portable and reusable application software becomes essential. The life cycle for COTS hardware, typically five years or less, is significantly shorter than the twenty plus years of an operational platform. Additionally, the development and maintenance costs related to software continue to increase in times of decreased funding.

Portable application software requires that the application software be independent of target processor and platform Operating Systems (OS) and that the coding of modules is standardized, such as ensuring ANSI C compliance. Any libraries referenced by the software must also be standardized and portable.

A variety of operating systems are available today for the target processors and platforms; however, no standard OS has been widely adopted by hardware vendors. POSIX seems to be current "standard" OS; however, POSIX is a "heavy" OS that contains many features that lead to a large memory requirements just for the OS. There are also real-time issues that have become the subject of debate. Because of the large amount of memory required even for a minimal POSIX implementation and the real-time issues, many vendors are reluctant (or refuse) to modify their OS to be POSIX compliant.

The high throughput requirements of signal processing applications require optimized libraries since most compilers generate code that executes slower by a factor normally in the range of four to six (or ever slower) than hand optimized code. Hardware vendors have through the years developed their own libraries of signal processing primitives which they optimize for the target processors that they support. These libraries, while similar in functionality, are not compatible. As a minimum, the calling sequences for the same functionality from two different vendors differ in the order parameters are referenced. Additionally, while the core functionality is the same, frequently a primitive from one vendor contains functionality that must be implemented by a sequence of two or more primitives from another vendor. As an example, an FFT might contain provisions for reordering the output.

There are, of course, different definitions (or more accurately levels) of "portability." True portability implies that the application can simply be recompiled for the new target/platform. In the case of the large applications under consideration, this might not be strictly true since the target/platforms are multiprocessor systems. As part of the port to a new target/platform it is likely that either repartitioning and/or reassigning sections of the application will be desired. Newer targets should have increased processing power and therefore the application can be executed on fewer processors. Since inter-processor communications (IPC) can lead to increased overhead, using fewer processors should result in increased efficiency by reducing the amount of IPC.

A less portable (or lower level of portability) implementation would have all Operating System interfaces, including inter-processor communications, isolated to a few modules. These modules would have to be modified for a port to a different OS.

An undesirable situation is to have OS and IPC mechanisms "sprinkled" throughout the application. Such an implementation becomes a nightmare when attempting to port to another target/platform.

When applications are ported to new targets/platforms, it is also likely that additional processing functionality will be added to the system. This might be in the form of modifications to some of the existing processing, or it might be the addition of new completely independent processing. A methodology for developing highly portable applications will permit both of these scenarios without extensive rework of the existing application software.

Reusable software must be target independent. Several levels of reusability should be included. At the highest level, applications should be easily incorporated into new or different platforms. As an example, the well developed algorithms for processing sonobuoys (DIFAR, DICASS, etc.) should be readily incorporated into platforms that are being developed primarily for new capabilities, such as dipping sonars. At an intermediate level, functionality that is commonly used in many applications, such as octave filtering, should be reusable in new applications. At the lowest level, a target independent specification of common processing blocks should be defined.

Portable, reusable application software should therefore have the following characteristics:

Operating System independence (or as a minimum, OS interfaces isolated to a few modules).

A methodology which permits a target independent specification of the processing, but, transparent to the user, provides links to libraries of optimized target specific processing functions.

A methodology which permits "easy" modification of the processing and does not require extensive hand rework.

A methodology which permits additional independent processing to be added to the application without extensive rework.

A methodology which permits repartitioning and/or reassigning sections of the processing without extensive rework.

The capability to incorporate reusable "blocks" defined at the "application," "subroutine," and "library module" levels.

Management Communications and Control, Inc. (MCCI) has developed a methodology and a toolset for developing and maintaining application software that is consistent with these characteristics, generating application software that is portable and reusable.

This document will describe the MCCI Autocoding Toolset and the associated portability and reuse methodology. Porting of applications which have been developed for the AN/UYS-2 will be described in considerable detail.

2. Application Overview

The MCCI Autocoding Toolset has as its foundation the Processing Graph Method (PGM). PGM is a Navy developed standard that can be used to specify signal processing (as well as some other types) applications using a data flow methodology. PGM implements the Karp and Miller data flow paradigm. This seminal work is the theoretical foundation for virtually all data flow methodologies. PGM is by far the most mature and critically evaluated of all data flow methods. Despite its close association with the AN/UYS-2, PGM has been maintained as a target independent data flow language and is well suited for specification of applications for COTS targets. PGM has both an iconic and a notational form.

2.1 Description of the Application

An application is specified as one or more PGM graphs, one or more Input/Output Procedures, and a Command Program. Each graph represents independent processing. The nodes in the application graphs specify the processing that is to be performed by that portion of the application. The nodes in the graph reference either a Domain Primitive or a user defined primitive that has been entered into the Autocoding Toolset as a "custom" Domain Primitive. Domain Primitives provide for target independent specification of the application. Since the graph has been specified using Domain Primitives, the graph has been termed the Domain Primitive Application Graph (DPAG). This term is used to distinguish this type of graph from other types of graphs that arise as part of the autocoding process. The MCCI Autocoding Toolset translates the PGM graphs into 'C' source code that incorporates calls to a vendor supplied target specific library of optimized signal processing functions.

Input/Output Procedures provide a mechanism for connecting the graph(s) to data sources and data sinks. For many target systems, the physical mechanism for this connection is custom i/o boards. Consequently, the user must manually code the Input/Output Procedures. A set of SRTS functions are provided which implement the interface with graphs. The user must use these SRTS services as part of every I/O Procedure.

The Command Program provides the mechanism for controlling the application. This typically involves an interface with some external device that for many applications includes an operator interface. Individual graphs may be started, stopped, and re-initialized. The values of variables that the graph(s) is using during execution may be viewed (read) or modified (written) by the Command Program. Data sources and sinks may be attached to (linked) or detached from (unlinked) individual graphs. The user

must construct the Command Program. A set of services is provided which implements the interface to graphs and to Input/Output Procedures.

A typical system is shown in Figure 1. Sensor data is processed according to the processing specified in the signal processing graphs. The results of the processing are typically shown on a display. An operator views the display and based on what is observed possibly modifies or otherwise controls the processing by sending messages to the Command Program. The Command Program translates operator commands into actions that configure and control the application. These actions are sent to the Graph Manager which modifies the application to conform to the desired processing.

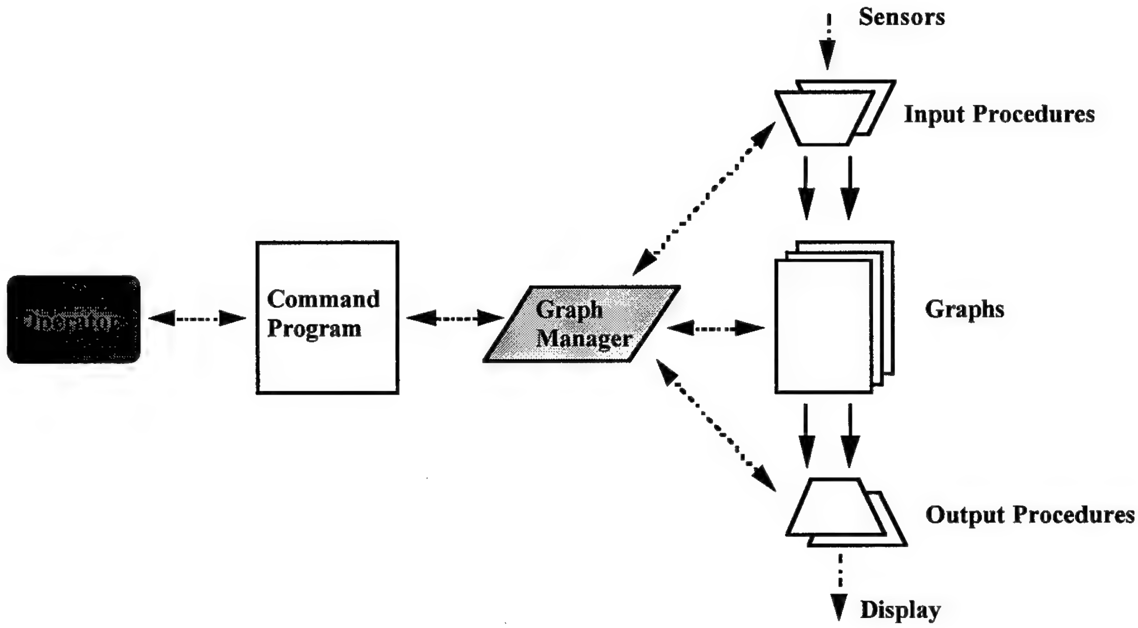


Figure 1. Typical System

The Autocoding Toolset includes Run-Time System components that provide services for graph execution, node scheduling, external control, queue management, and data transfer. Calls to these services are automatically inserted into the source code generated by the Autocoding Toolset.

2.2 Domain Primitives

The PGM graphs reference elements from an MCCI defined Domain Primitive Library as the primitives underlying the nodes of the graphs. The elements of the Domain Primitive Library are target independent kernel signal processing functions and data flow control specifications. Domain Primitives are intended to match the level of abstraction at which domain engineers design processes. The Domain Primitives provide support for all legitimate combinations of input and output data modes, structures, and multiple execution patterns. Domain Primitive Application Graphs (DPAGs) is the term used to identify PGM graphs utilizing Domain Primitives. Applications defined using DPAGs may be automatically translated to source code for

any supported COTS processors without modification of the DPAGs. Existing AN/UYS-2 graphs utilizing Q003 primitives may be easily converted to DPAGs. This conversion process is defined in Section 6.2 Converting AN/UYS-2 Graphs.

As part of the Autocoding Toolset translation process, each Domain Primitive is replaced by a sequence of one or more calls to elements of a vendor supplied library. This library of functions has been optimized for the specific target processor. (If the vendor does not provide an optimized library, a library of 'C' routines may be substituted.) The information required to translate the Domain Primitives to the sequence of vendor library elements is contained in Target Primitive Maps (TPMs) as shown in Figure 2. Primitive Library Organization. In order to port the Autocoding Toolset to a new vendor or to a new library, TPMs must be implemented by MCCI. Additionally, Primitive Library Units (PLUs) must be constructed for each vendor library element that is referenced by the set of TPMs. The primary information contained in PLUs is the execution time expression for each target processor type. This information is used in application execution simulation.

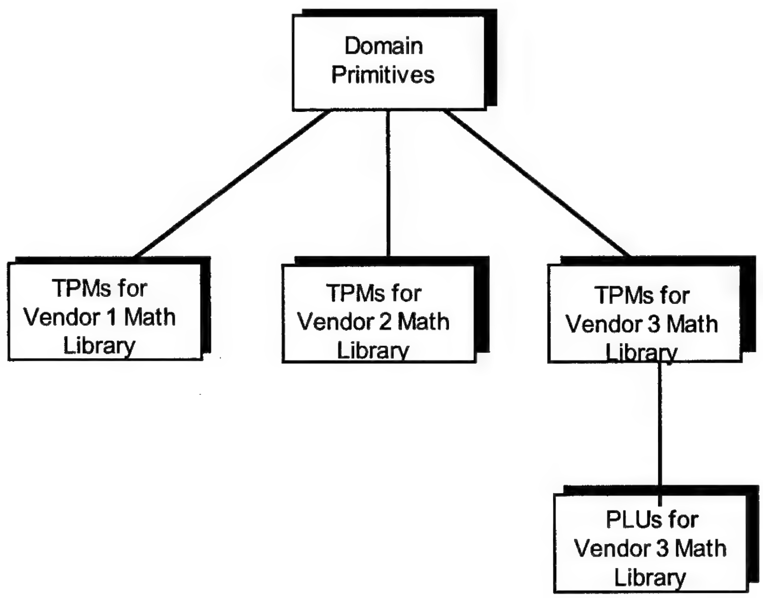


Figure 2. Primitive Library Organization

2.3 Control Programs

Command Programs are the control programs (written in a Higher Order Language or HOL) which configure/reconfigure the application based on events or external commands, typically generated by an operator. Command programs may be written in either 'C' or Ada or implemented as a graphical user interface (GUI). Calls to elements of a Command Program Interface Library (which is provided as part of the Autocoding Toolset) cause the Graph Manager component of the Run-Time System to invoke the appropriate action. Command Programs are dependent upon both the application and on the embedded host (particularly the host OS). The Command Program is reusable for control of the application executing on target platforms from different vendors, provided that the same host OS is used in the different systems.

3. Autocoding Toolset

3.1 Overview

The Autocoding Toolset developed by Management Communications and Control, Inc. (MCCI) is designed for large, complex signal processing applications that execute on multiprocessor platforms. Run-time support services are provided for reconfiguring and/or otherwise controlling the application and for supporting the execution of the application. The Autocoding Toolset starts from a target independent specification of the application and translates to a target dependent implementation. The target independent specification is easily ported to other targets by re-translating the application.

The MCCI Autocoding Toolset is used to translate signal processing applications that have been specified using the Processing Graph Method (PGM) into a set of 'C' language source code files that implement the signal processing functionality. The source code produced contains calls to functionality provided by the MCCI developed Static Run-Time System (SRTS). The SRTS implements graph management, graph execution, and queue management services which provide run-time support.

A high level diagram showing the components and some of the input required by the components of the Autocoding Toolset is shown in Figure 3. There are three tools that implement the core of the autocoding process. These are the Partition Builder, the MPID Generator (MPIDGen), and the Application Generator (AG). Also shown in the figure is the Static Run-Time System (SRTS) which provides run-time support services for graph execution and control and for queue (data) management. The SRTS is provided as a set of libraries. Calls to functions in these libraries that are required for graph execution are automatically generated as part of the autocoding process.

The Autocoding Toolset also contains a performance simulator, GSMP, which provides estimates of resource usage during execution of the application on the target hardware, a tool (Architecture Definition Tool) for generating a display of the hardware for use with the simulator and for generating a representation of architecture specific information for use by the core components of the Autocoding Toolset, and a tool (CP GUI) for generating sequences of commands that the Command Program would normally issue for use in testing applications.

3.2 Autocoding Process

The user partitions each application graph, determining which combination of nodes and subgraphs of the graph are to be grouped into a single schedulable entity. A partition will normally be a connected segment of a DPAG, but disjoint segments are permitted. The user provides the partitioning information using one of two methods which are described later. The user assigns each partition to a particular processor. More than one partition of a graph may be assigned to a particular processor. Partitions from different graphs may be assigned to a particular processor. The application is then ready for autocoding.

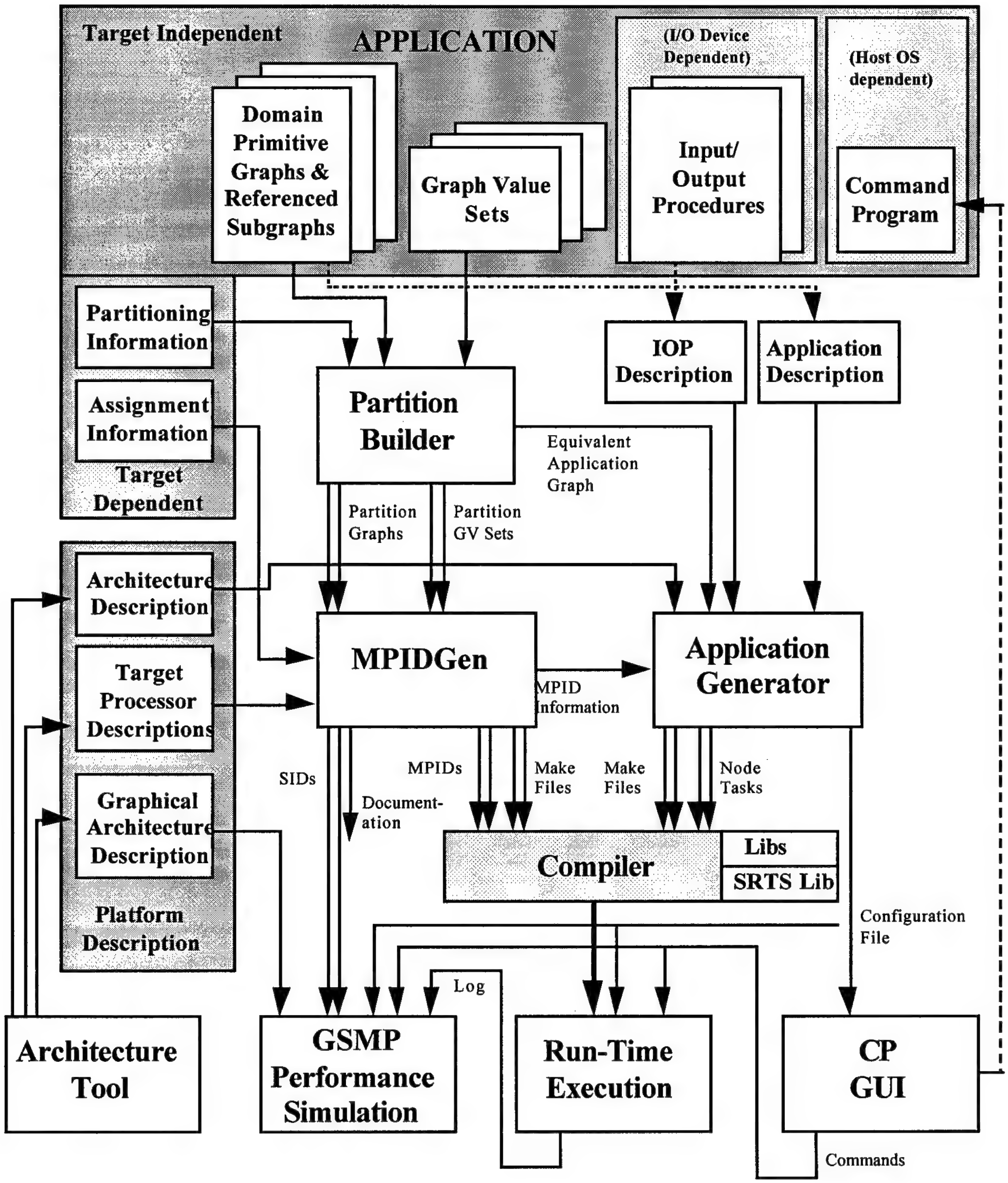


Figure 3. Diagram of the Autocoding Toolset

3.2.1 Partition Builder

The Partition Builder processes the partitioning information to form the partitions. A partition is a subset of the nodes of a DPAG that will become a single schedulable entity on a target processor. A target processor may have more than one partition assigned to it. A graph for each unique partition is generated by the Partition Builder. These graphs are called Partition Graphs.

An Equivalent Application Graph (EAG) is formed by replacing each partition in the DPAG with a single equivalent node that represents the partition. The primitive underlying each node in an EAG implements a control flow version of the processing for the partition represented by the node. The primitives are generated by the tool called MPIDGen from the Partition Graphs. MPIDGen is used in the next step of the autocoding process.

3.2.2 MPID Generator (MPIDGen)

Each partition graph is translated by MPIDGen into 'C' source code statements that implement a control flow version of the processing described by the partition graph. The partition graph is parsed to ensure a valid, error free, executable and translatable graph. Graph analysis translates the data flow graph to execution sequence(s) implementing graph transient and cyclic behavior for each set of enumerated control values. This analysis provides the specification for a control flow program implementing the MPID. A memory map is generated mapping queue operations into a set of fixed buffer addresses. The control flow version has been termed a MPID, an acronym for Multiprocessor Primitive Interface Description. This control flow implementation references primitives from a vendor library of signal processing functions for a particular target processor type such as an Intel i860 or a more standard DSP such as an Analog Devices 21060. Incorporating this type of library provides for efficient execution of the processing as these libraries have been optimized by the library vendor for the particular target processor. The source code generated by MPIDGen also includes calls to services provided by the Static Run-Time System for activities such as reading and writing queues.

A test utility executes the MPID as a single node application for unit testing on either a single target processor or the development workstation. Comparison of the processing results of the unit testing with corresponding results from an executable behavior model validates the autocoded translation.

3.2.3 Application Generator

When all of the Partition Graphs have been translated into MPIDs, the Equivalent Application Graphs (EAGs) for the entire application are translated by the Application Generator (AG) tool into 'C' source code and data structures that interface with the graph executing Static Run-Time System (SRTS). The Application Generator accesses the assignment information to assign each partition and therefore each equivalent node of each EAG to an actual processor. A node task wrapper is generated for each equivalent node (i.e., partition) that has been assigned to a

processor. This node task wrapper instantiates and calls the MPID function which was generated for the partition corresponding to the equivalent node.

In addition to a node task wrapper for each equivalent node, the Application Generator creates at least one thread manager for each processor in the architecture that has at least one partition assigned to it. Each thread manager maintains a list of equivalent nodes which have been assigned to the corresponding processor. The number of thread managers created for a processor is dependent upon which Operating System is used. For the MCOS implementation, a thread manager is created for each processor for each graph that has equivalent nodes assigned to that processor. At run-time, it is the thread manager task that actually creates the equivalent nodes associated with the graphs in the application.

The Application Generator also generates architecture specific files required by the Operating System and/or target specific cross-compiler. In addition, the Application Generator creates a file which specifies the application to the Graph Manager component of the SRTS.

3.2.4 Static Run-Time System

The Static Run-Time System (SRTS) consists of a Graph Manager and a set of graph execution support services. The Graph Manager provides the interface between the Command Program and the rest of the application. All application configuration messages from the Command Program are sent to the Graph Manager which processes the messages and invokes the processing associated with the message. The graph execution support services provide initialization functions, queue data management services, node scheduling services, and services for communication with the Graph Manager. Calls to the services are embedded into the source code generated by the autocoding process.

3.3 Ancillary Support Tools

3.3.1 Command Program Graphical User Interface

The Command Program Graphical User Interface (CP GUI) provides the user with an easy to use Command Program interface. The CP GUI can be used to control complete applications or portions of the application without having to construct a Command Program. Since the CP GUI implements all of the application control functions of the Command Program using a menu interface, it can be especially useful during the development and unit testing phase when the interface to the application may be changing. With the CP GUI, the user can issue single commands or construct sequences of commands as macros. The macros form the basis for generating either components of the final Command Program or Command Program scripts which can be interfaced to a custom GUI.

3.3.2 Performance Simulator

The performance simulator, GSMP, uses a model of the hardware, a model of the Static Run-Time System, models of the partitions (generated by MPIDGen), and a

description of the application (generated by Application Generator) to estimate the resource usage encountered during the execution of the application on the target platform. During simulation, resource usage is visible via a display. Many resource usage problems are easy to detect by watching the simulation. A statistics report is also generated. Control of the application for GSMP simulation is via the CP GUI, and macros generated by the CP GUI for test purposes can be reused, or macros developed for simulation can be reused during test. Additionally, GSMP can playback logs created during execution of the application on the actual target hardware, providing a high degree of visibility into application execution.

3.3.3 Architecture Definition Tool

The Architecture Definition Tool permits the user to define or modify target processor types, define the target architecture in a format compatible with the Autocoding Toolset, and define a graphical view of the target architecture for use with GSMP.

3.3.4 Graph Translation Tool (GrTT)

An executable Ada partition behavior model may be automatically generated for each partition using GrTT. Behavior models will exhibit step-by-step execution behavior that is identical to the autocoded partition with numerical processing results that may be compared to corresponding queue contents in the executable architecture graph. Test vectors for validation on the target architecture may be generated. GrTT test vectors can be used to verify design requirements capture and to validate partition autocoded programs.

3.3.5 Virtual Design Machine (VDM)

The UNIX based network target planned for SBIR N94-165 Phase III will serve as a high capacity functional simulator as well as an operational target.

4. Productivity

Utilizing MCCI's Autocoding Toolset to develop applications enables users to realize an order of magnitude reduction in software development cost and a four fold decrease in development times. Code with run-time performance that is comparable to hand generated code is produced. A refereed evaluation of the productivity enhancement our tools provide was conducted by MIT Lincoln Laboratory as part of the RASSP program.

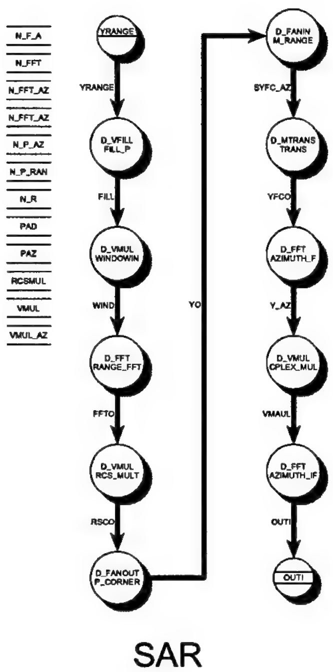
4.1 Benchmarking of the MCCI Autocoding Toolset

The RASSP program demonstrated its goal of improving embedded signal processor development productivity by a factor of four through benchmarking MCCI's Autocoding Toolset. A Synthetic Aperture Radar (SAR) signal processing algorithm developed by MIT Lincoln Laboratory was used for tool and methodology testing. The benchmark was initially implemented using traditional methods to establish a productivity baseline. Productivity enhancements realized with the Autocoding Toolset were formally evaluated by Lincoln Laboratory and compared with the baseline. MCCI also

used the benchmark algorithm to demonstrate performance of the GrTT behavior modeling tool developed under the RASSP technology base effort.

The SAR benchmark software allocation was implemented with an alpha version of the Autocoding Toolset.

Figure 4 shows the PGM DPAG. Figure 5 shows graphs of the range and azimuth partitions and the equivalent application graph created by the Partition Builder. Each unique MPID was autocoded and unit tested by comparing its results to test vectors generated by the behavior model and algorithm simulation tools. Figure 6 shows a comparison of behavior model (GrTT) output for the range partition graph and the corresponding vector from the MPID unit test.



The SAR benchmark processing requirements were allocated to hardware processing requirements and a software architecture. The software allocation included range and azimuth processing separated by a corner turn. SAR images were formed from 1K range returns of 2K complex words each. The required frame rate is one second, requiring support of 2 MHz complex word input and output rates. A latency constraint of three seconds was also required. Range processing transformed each range return into a complex spectrum sorting return by bearing dependent Doppler. Corner turning transposed the range processed data into bearing range alignment. Azimuth processing convolved the range returns for each bearing with a Doppler/range compensation kernel. The processing load was approximately 500 Mflops.

Figure 4. PGM Domain Primitive Application Graph for SAR Benchmark

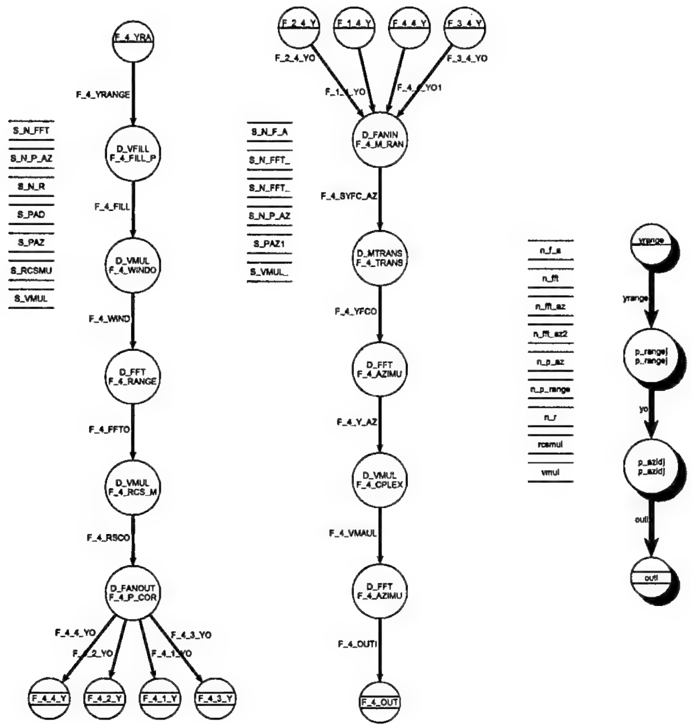


Figure 5. Range and Azimuth Partition Graphs and the Equivalent Application Graph for SAR Benchmark

Partition graphs are autocoded into executable programs encapsulated in nodes of the equivalent application graph. The equivalent application graph is autocoded into the run-time image of the application.

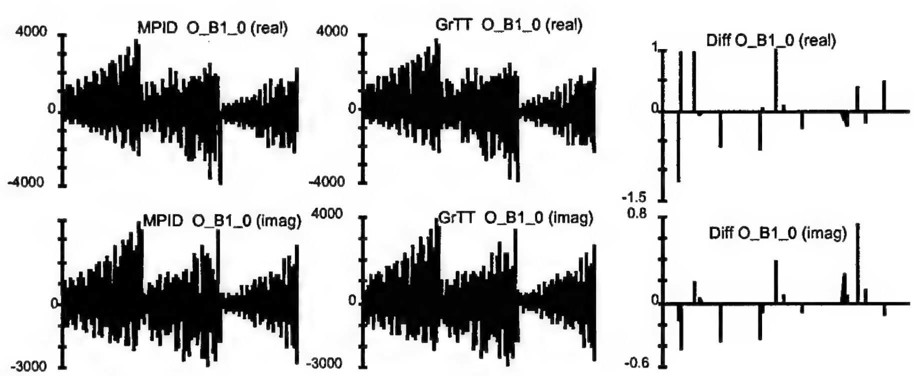


Figure 6. Comparison of GrTT Ada Behavior Model and MPID Unit Test Output Vectors

Processing performance was comparable to the hand generated baseline. Significant productivity improvements were demonstrated. Table 1 lists the comparisons between the handcoded baseline and the results of autocoding. Test results show comparable computational performance between baseline and autocoded implementations and significant productivity enhancements. A factor of ten was achieved in reducing development time including the time spent iterating the design. The development time recorded with the autocoding tools includes the tool use learning curve and the several design iterations. Significant further productivity improvements are expected with the commercial release version of the Autocoding Toolset.

Measured Item	Hand-Coded	Autocoded	Comment
Lines of Code	2361	3855	a. user must generate SPGN for Domain Primitive Graph b. RTS not included, MYA port c. IOP not included (500 LOC)
Perform-ance	<7 sec/sec 7 i860 Nodes	6.85 sec/sec 8 i860 Nodes	a. 8 Nodes needed for memory b. Measured loading supports 7 node partitioning
Memory	32M	85.5M 29.5M (6/96)	a. Autocoding tool limitation b. Upgrade in beta version
Develop-ment Time	8 MM	0.75 MM	a. 10 X improvement b. includes learning time - should improve in future releases
Test Time	2.5 MM	0.5 MM	a. 5 X improvement

Table 1. Comparison of Autocoding with Handcoding

4.2 MIT Lincoln Laboratory's Software Cost Model

The reduction in cost and schedule impacts of the productivity improvements demonstrated are illustrated in Figure 7. These charts have been excerpted from the viewgraph presentation, "Modeling RASSP Benefits," of an independent study of RASSP productivity improvements by Dr. James C. Anderson of MIT Lincoln Laboratory. Time and cost data measured during RASSP benchmarking were analyzed using COCOMO (Constructive Cost Model) and REVIC (Revised Intermediate COCOMO) to develop the comparison of developing a large real-time processing software system using RASSP HW/SW codesign methodology (PGM based executable requirements and tools, virtual prototyping, and autocoding) with the standard six phase development program. The cost model results are dramatic; a 3.5 reduction in schedule and 7.4 reduction in cost are predicted. If the sponsor provides executable requirements in the form of reusable graphs, the model predicts a factor of 9.09 reduction in cost and a factor of 6.25 reduction in schedule. This latter case typifies life cycle maintenance and P^3I insertion of new processing technology in the Autocoding Toolset.

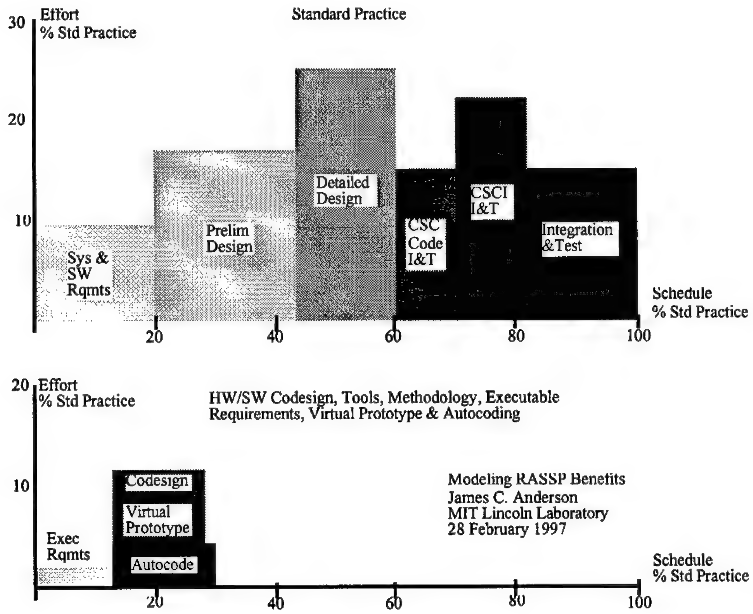


Figure 7. Cost and Schedule Comparison of Software Development Using RASSP PGM Based HW/SW Codesign Methodology and Tools vs. Standard Practice

5. Portability and Reusability

Domain Primitive Application Graphs may be reused on any target platform that is supported by the MCCI Autocoding Toolset. Reuse involves specification of the new hardware architecture, possibly repartitioning of the application graphs for the new target, and automated generation of the application. New hardware architectures may be technology upgrades of a vendor's boards or designs using boards from a different vendor. HOL command programs may be reused on new hosts for which target platform OS support or interface exists. Reuse capability makes the graphical application specifications and HOL control programs become the high value, reuse software. Reuse will also minimize program dependence on any particular hardware vendor. The existing base of AN/UYS-2 graphical applications may be readily added to the reuse library after porting the graphs to be Domain Primitive Graphs, making them usable on all supported hardware targets. Ease of reuse will radically reduce software life cycle costs.

5.1 Porting the Autocoding Toolset to New Target Platforms

Porting of the Autocoding Toolset requires effort at several levels. First, the Target Primitive Maps (discussed in Section 2.2 Domain Primitives) must be implemented whenever a new vendor supplied library is to be incorporated into the Autocoding Toolset. The API to a particular vendor's library is usually not dependent upon the type of target processor since vendors are concerned with compatibility of legacy code. However, the API is normally different. Second, execution time estimates for elements of the vendor supplied library must be entered in PLUs (discussed in Section 2.2 Domain Primitives) in order to simulate applications. Execution time estimates are dependent upon target processor type (e.g., PowerPC, SHARC, etc.). Finally, the

Static Run-Time System (SRTS) must be modified to use elements of the target platform multiprocessor Operating System. Mechanisms for messaging, semaphoring, shared memory constructs, and data transfers are dependent upon the facilities of the OS. The changes required for the SRTS are isolated to a low level, comparable to device drivers.

MCCI has been following the ARPA sponsored VSIP program. VSIP is developing a specification for a library of vector, signal processing, and imaging functions that in many respects parallels the Domain Primitive Library. VSIP is also implementing a reference version of the library and performance versions of a subset of the library. It is a goal of VSIP to have vendors implement versions of VSIP that have been optimized for their target processors and platforms.

Incorporating the VSIP library, including the vendor optimized target specific versions, into the Autocoding Toolset can be achieved using an organization shown in Figure 8. The Domain Primitive Library would then support, without any modifications, any target processor that had a VSIP compliant library. Some effort would be required to incorporate the execution time expressions necessary for performance simulation. If timing expressions were provided, this effort would be on the order of one week.

Under a separate program, MCCI has been investigating the impact of incorporating VSIP into the Autocoding Toolset and the impact VSIP would have on execution efficiency. The VSIP API uses an object oriented approach with data being represented as views as opposed to the standard 'C' approach of memory locations. The object oriented approach does add overhead, in both increased execution time and increased memory usage. The overhead is dependent upon the application; however, it is also based on the particular VSIP implementation being used. Preliminary measurements indicate that the overhead should be tolerable for most systems.

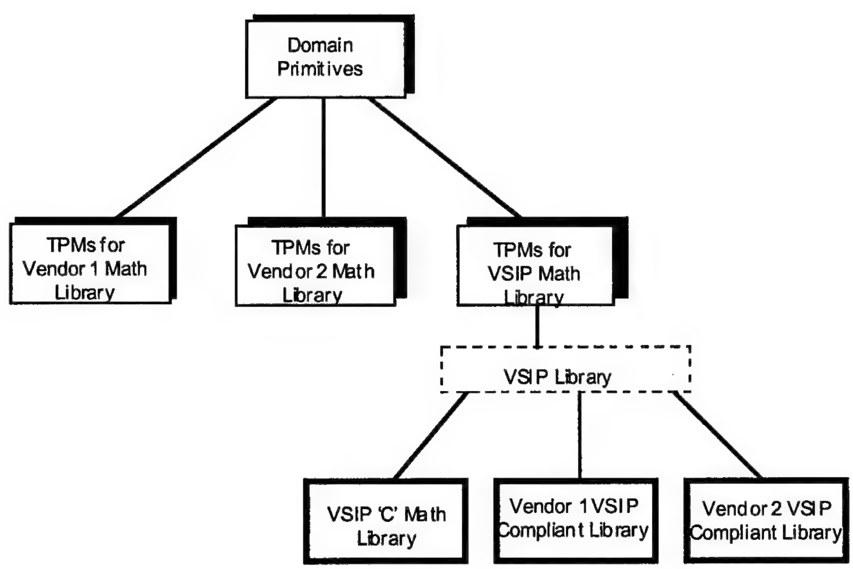


Figure 8. Primitive Library Organization Extended for VSIP

The importance of having target specific optimized libraries cannot be overstated. It is these modules that provide the core of high throughput. Most current compilers do well to provide executables that are three to four times slower than the optimized library elements. This is deemed unacceptable, since it translates into three or four times more hardware with the associated increases in cost, weight, power, space, and maintenance, and the decreased reliability.

5.2 Porting the Run-Time System to New Target Platforms

The MCCI graph executing Static Run-Time System (SRTS) implements a Graph Manager, which interfaces to the Command program for external control of the application, and a set of services which provides for queue and data management and for determination of when a node is ready for execution. The SRTS implements a standardized Application Program Interface (API) and calls to the services are embedded into the application specific source code generated by the Autocoding Toolset. The SRTS services interface with the underlying Operating System for task scheduling, Inter-Process Communication, and other operations normally associated with OS services.

In order to understand the MCCI Operating System requirements, one must first understand the hardware model. In the "normal" hardware configuration, there is an embedded host processor and one or more groups of "signal processors." (It is possible to configure the system such that a host is not required.) Each Group of "signal processors" can consist of one or more processor boards typically consisting of 16 or more processors. An application consists of one or more signal processing graphs, I/O Procedures, and a Command Program. A graph resides entirely within a Group. Data from a graph can be piped to another graph. The graph receiving the data may be located in the same Group or in another Group.

The MCCI graph executing Run-Time System requires minimal OS support. The OS must span all processors within a Group. The expected services are:

a. Process/Thread Scheduling (including priority preemption, if possible). The process/thread scheduling may be two level (such as Mercury provides with both "process" and POSIX thread support) or a single task level (such as SPOX provides).

b. Semaphores for signaling/synchronizing both locally and within the Group. Both blocking and non-blocking access functions are required. Time-outs are highly desirable.

c. Messaging (mailbox or socket) both processor local and within the Group. Both blocking and non-blocking functions are required. Time-outs are highly desirable.

d. Data transfer routines for reading and for writing data locally and within the Group. The writing services must include provisions to block until the transfer has completed and the data has been stored in the memory location(s). These services should use the quickest transfer mechanism available. If the architecture supports a shared

memory model, these routines can be rather simple. If the architecture does not support a shared memory model, these routines become more complex.

e. Dynamic memory allocation functions (alloc and free) must be provided.

f. If SHARCs are the target processor, it is highly desirable that code overlay support be provided by both the OS and the compiler/linker. This is due to the limited on-chip memory and the fact that there is a single off-chip bus.

There must be a method of messaging that exists between the host and at least one processor within each Group. This messaging may be socket or mailbox. For example, with the Mercury hardware, MCOS sockets are used for this type of communications. Mercury provides the MCOS drivers for a variety of Sparc boards executing Solaris.

There must also be a mechanism for sending data to and receiving data from any I/O boards that interface with the external world. For example, Mercury provides services to interface boards with the Raceway.

A run-time loading capability is highly desirable. This permits loading of new tasks during run-time reconfiguration. If this is not provided, then the load image for each processor must contain all code that can be executed on that processor for all configurations of that application. Some loader must be supplied either as part of the OS or as part of the Board Support Package.

There must be some method of accessing information so that the SRTS (during MCCI porting to the platform) and applications can be debugged. It would be nice to have stdio available from the signal processors. If stdio is not available from the signal processors, file i/o and/or real-time printf capability from the signal processors would prove very useful. As a minimum, post-mortem printf (e.g., from a trace buffer) must be provided.

MCCI has been following the Navy sponsored Common Operating Environment (COE) with interest. If a common OS API could be defined, the SRTS could be modified to the API and not have to ported for each platform. Additionally, under a separate program, MCCI will be investigating a MPI compliant interface. Assuming that an efficient MPI compliant version of the SRTS can be developed, SRTS ports would not be required for platforms that had MPI capability.

5.3 Reusable Domain Primitive Application Graphs

The reuse strategy is based on the portability of Domain Primitive Application Graphs, DPAGs. DPAGs are completely target independent PGM data flow graph specifications of signal and data processes. The middleware interfaces to target specific computational routines incorporated in the Autocoding Toolset and the run-time interfaces to target operating systems make it possible to generate executable code implementing the DPAG specifications on all family targets. If properly exploited,

this reuse capability can profoundly affect acquisition and life cycle support strategies for airborne signal processing systems to the advantage of the Navy.

5.4 HOL Control Program Reuse

The HOL control programs, programs invoking PGM command procedures with calls to elements of the Command Program Interface Library, may be readily reused. The Command Procedure Interface Library is available in 'C', and could be readily extended for a version with the routines encapsulated in Ada. Command programs are specific to the application and possibly the operating system of the embedded host. Provided the formal inputs to the application are not changed in porting it to a new target, the control program will be reusable with the new target if the same host OS is used. Control programs themselves may be ported to a new host provided the target interface support exists. This support is some means of messaging between the host and the target (typically using a form of sockets).

6. Reuse of Existing AN/UYS-2 Applications

The Autocoding Toolset reuse capability offers the opportunity to reuse the $100M plus AN/UYS-2 code base of PGM application graphs at minimal costs.

6.1 AN/UYS-2 Command Programs

The core functionality of a Command Program is to translate commands from an external source such as an operator console into actions that configure/reconfigure the application.

While the potential to convert AN/UYS-2 Command Programs exists, there are many issues involved. The first is that AN/UYS-2 Command Programs were written in Ada for execution on a 68030 processor. The cross compiler that was used is no longer supported. Porting Ada to new targets is not as simple as a recompile.

Additionally, while both the AN/UYS-2 and the MCCI Autocoding Toolset are based on PGM, the implementations have differences. The AN/UYS-2 implementation requires the Command Program to perform system level operations such as creating mailboxes for communications with Input Output Procedures (IOPs). The AN/UYS-2 implements IOPs as Graph Support Programs. Each Graph Support Program must communicate with the Command Program. The MCCI implementation requires communication with the Graph Manager component of the SRTS only. Also, the AN/UYS-2 implementation uses a concept called Graph Support Nodes for access to queue or graph variable data by the Command Program or IOP. Graph Support Nodes are not defined in the PGM Specification, and they are not used or supported by MCCI.

Thus, if it is desired to attempt to reuse AN/UYS-2 Command Programs, two porting issues exist. (1) The Command Program will have to be recompiled using the GNAT or some other Ada compiler. Compiler differences will have to be resolved using standard debugging and code revision procedures. (2) Calls to the AN/UYS-2 GrM interface must be replaced with equivalent calls to the Command Program Interface Library elements. For the most part, this is a straightforward substitution of

semantically identical procedure calls. There is some Command Program functionality introduced into the AT&T implementation that is not supported in the PGM specification typically dealing with Graph Support Nodes and mailboxes. MCCI could add support for these as required so that Command Program Interface Library call substitution may be used for all AN/UYS-2 GrM interface functions. Some of the new support functions would be simple stubs, others would be fairly complex.

6.2 Converting AN/UYS-2 Graphs

AN/UYS-2 graphs may be readily converted into DPAGs with simple graph editing. Node statements for Q003 primitives must be edited into node statements for Domain Primitives. In general, there is a many to one mapping of Q003 primitives into Domain Primitives. To anyone familiar with AN/UYS-2 programming, the transformation will be intuitive. The editing may be accomplished with the DSPGraph Tool or with a simple text editor. Once converted to DPAGs, the AN/UYS-2 graphs may be autocoded for any architecture that the Autocoding Toolset supports.

MCCI converted the AN/UYS-2 DICASS graph to a DPAG implementation as part of this project. The top level graph is shown in Figure 9. The expanded graph contains on the order of 609 nodes. The conversion results are described in Section 6.4 DICASS Conversion. Based on the conversion process performed on this application, AN/UYS-2 graphs and referenced subgraphs can be readily converted to Domain Primitive Graphs by performing the following steps:

1. Convert the primitive referenced (`PRIMITIVE =`) by each `%NODE` statement from a Q003 primitive name to a Domain Primitive name. A cross reference table of Q003 Primitives to Domain Primitives can be found in Appendix A. For some conversions, the parameter lists do not match. A cross reference table containing the parameter lists can be found in Appendix B. For some primitives, more than one Domain Primitive can be selected. To select the “proper” one, the user should understand the functionality of the Q003 primitive in the context it is being used and should understand the functionality of each of the Domain Primitive choices.

If a node in the Q003 graph references a chain, there will be no Domain Primitive equivalent. Instead, the procedure of the next section should be followed.

2. If there is no Domain Primitive equivalent for the Q003 primitive, there are two options. A request can be sent to MCCI to add a new Domain Primitive. Alternatively, one can construct a ‘C’ procedure and encapsulate it as a “Custom” Domain Primitive as described in the user manuals for the MCCI Autocoding Toolset.

3. The AN/UYS-2 uses a 16 bit representation for single precision and a 32 bit representation for double precision. Most newer targets use 32 bit for single precision and 64 bit for double precision. Additionally, many newer targets do not have hardware support for double precision (64 bit) and only provide software emulation which does not execute quickly. Converting AN/UYS-2 double precision to 32 bit single precision should therefore be performed by modifying the mode declarations in the graph (e.g., `DFLOAT` is modified to `FLOAT`).

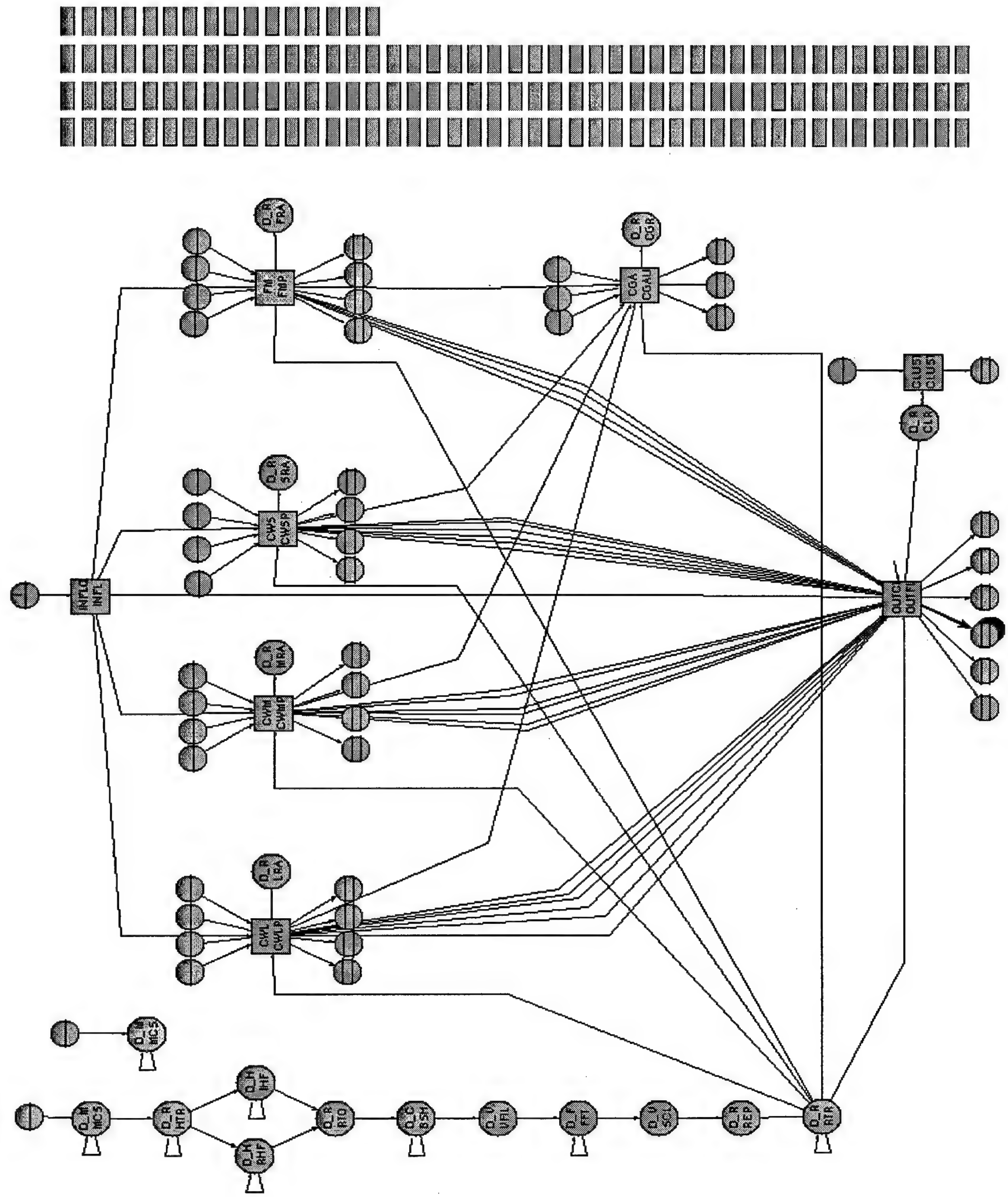


Figure 9. DICASS Graph

6.3 AN/UYS-2 Chains

Chains written for the AN/UYS-2 require special treatment to port them to the architecture family. Two approaches are possible; one is to reverse engineer the chain from the PID language implementation, and the other is to use the graphical representation, if it exists.

Chains for which there is no equivalent graphical specification may be handled in one of several approaches. (1) A graphical specification may be reverse engineered from the PID language. MCCI reverse engineered several graphical representation of ALFS PIDs during the PIDGen program. Once a graph is obtained, the chain may be handled as described above. (2) The PID language program may be rewritten as a 'C' program. Target specific math library calls may be substituted for E002 microcoded procedure calls. Since PID language is a subset of 'C', this may be the easiest path for chains for which graphical specifications do not exist. A planned encapsulation tool will provide for their incorporation in the Domain Primitive Library as user primitives.

A graphical representation of the chain's processing may be entered into the reuse library as DPAGs. Subgraphs referring to the chain's DPAG may be substituted for the chain node in the top level graphs. For P3UIV chains that are specified in graphical format, this is the preferred approach. If the graphical format is not available, it may be possible to locate documentation containing a pictorial representation from which the graph can be reconstructed. NEPs and modes must be added to the iconic representation in the P3UIV chain specification to make them complete DPAGs and nodes must be converted to reference Domain Primitives. This entire procedure is a relatively straightforward task as detailed below.

Some, if not all, existing AN/UYS-2 applications, contain one or more chains that will have to be converted for inclusion in the ported application. A chain was constructed from a segment of a graph. Chains increase the execution efficiency by increasing the processing performed by a schedulable entity (i.e., the node). In many respects, chains and MPIDs are similar.

In order to convert a chain for use with the MCCI implementation, the following process is suggested. After the process has been described, an example will be shown. It is assumed that the reader (and most definitely the person doing the conversion) is familiar with PGM.

6.3.1 Creating the Graph from the Chain Description

The first two steps are to 1) create a graph containing nodes with underlying Q003 primitives and 2) convert the "Q003" graph to a "Domain Primitive" graph.

1. Beginning with the description of the chain, construct a graph using Q003 primitives. This is a relatively straightforward procedure using the written description of the chain and the pictorial pseudo-graph diagram given in the description. The diagram should contain each of the nodes, the `PRIM_IN` and `PRIM_OUT` lists for each

node, and the queue connectivity between nodes. The Node Execution Parameters (threshold, read, offset, and consume) are not given, and they must be inferred from the processing of the chain.

Additionally, the graph header must be derived. Determining the formal parameters is made by examining the PRIM_IN and PRIM_OUT lists for the chain. However, there is a distinction between a %NODE statement description and a %GRAPH description. The %NODE statements have PRIM_IN and PRIM_OUT lists, while %SUBGRAPH statements have GIPs, VARs, INPUTQs, and OUTPUTQs. Determining which of the PRIM_IN and PRIM_OUT parameters are the formal input and output queues is normally straightforward. More difficult is determining which of the other PRIM_IN and PRIM_OUT parameters should be GIPs and which should be VARs. At this point, a best guess based on usage of the parameter is suitable, as the determination will be revisited in a later step in the conversion process.

2. Convert the graph from one using Q003 primitives to one using Domain Primitives. This procedure is straightforward and is detailed elsewhere in this document. (See Section 6.2 Converting AN/UYS-2 Graphs.)

6.3.2 Modifications Based on Application Specific Use

The next steps pertain to modifying the calling graph and to incorporating application specific usage information into the graph implementing the chain.

3. Modify the calling graph by replacing the node statement with a subgraph statement. This step can be tricky on occasion due to the mismatch between %NODE statements and %SUBGRAPH statements noted previously. For the moment, ignore any PIP_IN and PIP_OUT statements. Using the graph from step 2 as a template in conjunction with the actual usage (i.e., GIPs and VARs) of entities in the %NODE statement, create the SUBGRAPH statement. Modify the graph of step 2 as appropriate, declaring the entities as either GIPs or VARs.

4. Next, any application specific usage related information must be incorporated. This includes accounting for any PIP_INs and/or PIP_OUTs, valves and expressions that are part of the %NODE statement. The ASNP chain example described below contains these types of application specific information and describes how the information can be incorporated.

6.3.3 CHN_ASNP Example

The conversion process for CHN_ASNP as used by the CWASCAN graph in the DICASS application is shown as an example. The Chain Description is contained in Appendix A. From this description, the Q003 graph implementing the chain is developed.

Graph Body:

The Graph Body is constructed directly from the chain description. First declare the local queues. This information is taken directly from the chain description.

```
%%  LOCAL QUEUE LIST FROM CHAIN DESCRIPTION
%QUEUE( X1 : DCFLOAT )
%QUEUE( X2 : DCFLOAT )
%QUEUE( X3 : DCFLOAT )
%QUEUE( X4 : DCFLOAT )
%QUEUE( X5 : DFLOAT )
%QUEUE( X6 : DCFLOAT )
%QUEUE( X7 : DCFLOAT )
%QUEUE( X8 : DCFLOAT )
%QUEUE( X9 : DFLOAT )
%QUEUE( X10 : DFLOAT )
%QUEUE( X11 : INT )
```

Next construct the node statements, ignoring Node Execution Parameter (`NEP`) values, and declaring variables (`GIPs` or `VARs`) as needed. Most of this information is taken directly from the chain description figure by referring to the tables associated with each node. Construction of two nodes, one referring to Q003 primitive `DFC_FCTR` and one referring to Q003 primitive `DCP_SPL` is shown. A node name must be assigned to each node. This name can be anything the user wishes, but each name must be unique. For some parameters, it is useful to construct a variable. In the second node, the fourth parameter, which is an array `{FFTSZ, 1}`, will be replaced by parameter `SPL1_BLS`.

```
%NODE  ( FCTR
        PRIMITIVE = DFC_FCTR
        PRIM_IN   = DASC*FFTSZ,
                    2,
                    1,
                    BB,
                    FAMILY[OMNI, CRD]
                       THRESHOLD = ???
        PRIM_OUT  = FAMILY[X1]
)

%NODE( SPL1
   PRIMITIVE = DCP_SPL
   PIP_IN    = ASNP_VALVE
   PRIM_IN   = DASC*FFTSZ,
               1,
               SPL1_BLS,
               X1   THRESHOLD = ???
   PRIM_OUT  = FAMILY[X2] VARIABLE VALVE = ASNP_VALVE
   )

%% Need to declare variable SPL1_BLS
%GIP( SPL1_BLS : INT ARRAY(2) INITIALIZE TO {FFTSZ, 1} )
```

The other `%NODE` statements are constructed in a similar fashion.

Graph Header

The Graph Header is constructed next. The information for this step is contained in the Parameter List, augmented by the information in the Parameter Table. The Graph Name and the `INPUTQ` and `OUTPUTQ` lists are usually easy to construct.

```
%GRAPH( ASNP
   INPUTQ  = MEF  : DFLOAT,
             OMNI : DCFLOAT,
             CARD : DCFLOAT
   OUTPUTQ = ASOT : INT V_ARRAY(KK) )
```

The `GIP` and `VAR` lists can be tricky, in that it is sometimes hard to determine if a parameter should be a start-time parameter (GIP) or run-time parameter (VAR). At this point, a best guess only is required. The lists will be revisited during a subsequent step.

```
   GIP      =
             DASC     : INT,
             NAS      : INT,
             NS       : INT,
             FFTSZ    : INT,
             NIF      : INT,
             NFSS     : INT,
             BB       : INT,
             ASWIND   : INT ARRAY(2),
             REQ      : DFLOAT ARRAY(6),
             MNA      : INT ARRAY(2),
             DM1      : INT,
             DM2      : INT,
             %% V Array Size on output queue
             %% nominally max will be KK = (NS*NFSS)/DASC + 8
             %%
             KK       : INT
   VAR      = ASNP_VALVE : INT,
             ASGN     : DFLOAT,
             HEADER   : INT ARRAY(8)
```

Note that a parameter, `KK`, for the maximum size of the v_array output queue was added to the parameter list. This is to avoid hard coding the size into the `OUTPUTQ` declaration.

At this point, the `NEP` values are still required. From the Chain Description, it is seen that multiple execution depends upon the expression: `NE = NS/DASC`.
The nominal input data amount into `DFC_FCTR` is `DASC*FFTSZ`. If we multiply this amount by `NE`, we obtain the threshold amount for `INPUTQ`s `OMNI` and `CARD`., namely `NS*FFTSZ`. The `NEP`s for the other nodes in the graph can be similarly derived.

Putting together all the pieces, we obtain the Q003 Primitive graph:

```
%GRAPH( ASNP   %%%%%%              CHN_ASNP       Q003 version
   GIP      =
             DASC     : INT,
             NAS      : INT,
             NS       : INT,
             FFTSZ    : INT,
             NIF      : INT,
             NFSS     : INT,
```

```
            BB       : INT,
            ASWIND   : INT ARRAY(2),
            REQ      : DFLOAT ARRAY(6),
            MNA      : INT ARRAY(2),
            DM1      : INT,
            DM2      : INT,
            KK       : INT                   %% V Array Size on output queue
                                             %% nominally max will be KK =
                                             %% (NS*NFSS)/DASC + 8
   VAR      = ASNP_VALVE: INT,
            ASGN     : DFLOAT,
            HEADER   : INT ARRAY(8)
   INPUTQ   = MEF      : DFLOAT,
            OMNI     : DCFLOAT,
            CARD     : DCFLOAT
   OUTPUTQ = ASOT     : INT V_ARRAY(KK)
  )

%GIP( SPL1_BLS : INT ARRAY(2) INITIALIZE TO {FFTSZ, 1} )
%GIP( SPL2_BLS : INT ARRAY(2) INITIALIZE TO {NAS, 1} )

%QUEUE( X1 : DCFLOAT )
%QUEUE( X2 : DCFLOAT )
%QUEUE( X3 : DCFLOAT )
%QUEUE( X4 : DCFLOAT )
%QUEUE( X5 : DFLOAT )
%QUEUE( X6 : DCFLOAT )
%QUEUE( X7 : DCFLOAT )
%QUEUE( X8 : DCFLOAT )
%QUEUE( X9 : DFLOAT )
%QUEUE( X10 : DFLOAT )
%QUEUE( X11 : INT )

%NODE  ( FCTR
        PRIMITIVE = DFC_FCTR
        PRIM_IN   = DASC*FFTSZ,
                    2,
                    1,
                    BB,
                    FAMILY[OMNI, CRD]
                       THRESHOLD = NS*FFTSZ
        PRIM_OUT  = FAMILY[X1]
)

%NODE( SPL1
   PRIMITIVE = DCP_SPL
   PIP_IN    = ASNP_VALVE
   PRIM_IN   = DASC*FFTSZ,
               1,
               SPL1_BLS,
               X1  THRESHOLD = NS*FFTSZ
   PRIM_OUT  = FAMILY[X2] VARIABLE VALVE = ASNP_VALVE
   )

%NODE( SPL2
   PRIMITIVE = DCP_SPL
   PIP_IN    = ASNP_VALVE
   PRIM_IN   = DASC*NAS,
               1,
```

```
                SPL2_BLS,
                MEF   THRESHOLD = NS*NAS
   PRIM_OUT   = FAMILY[X5] VARIABLE VALVE = ASNP_VALVE
   )

%NODE( REORD
   PRIMITIVE = DFC_REORD
   PRIM_IN   = FFTSZ,
               FFTSZ,
               1,
               FFTSZ/2+1,
               FFTSZ/2+2,
               FFTSZ
               X2  THRESHOLD = NS*FFTSZ/DASC
   PRIM_OUT  = X3
   )

%NODE( SPL3
   PRIMITIVE = DCP_SPL
   PRIM_IN   = FFTSZ,
               1,
               ASWIND,
               X3  THRESHOLD = NS*FFTSZ/DASC
   PRIM_OUT  = FAMILY[X4]
   )

%NODE( MUL
   PRIMITIVE = VRC_MUL
   PRIM_IN   = NAS*NS/DASC,
               1,
               X5  THRESHOLD = NAS*NS/DASC,
               X4  THRESHOLD = NS*FFTSZ/DASC
   PRIM_OUT  = X6
   )

%NODE( REORD2
   PRIMITIVE = DFC_REORD
   PRIM_IN   = NAS,
               NIF,
               1,
               (NAS+1)/2,
               (NAS+1)/2,
               NAS,
               X6  THRESHOLD = NAS*NS/DASC
   PRIM_OUT  = X7
   )

%NODE( FFT
   PRIMITIVE = FFT_CC
   PRIM_IN   = NIF,
               NFSS,
               1,
               (NIF-NFSS)/2 +1,
               X7  THRESHOLD = NIF*NS/DASC
   PRIM_OUT  = X8
   )

%NODE( PWR
   PRIMITIVE = VOC_PWR
   PRIM_IN   = NFSS,
```

```
              X8   THRESHOLD = NFSS*NS/DASC
   PRIM_OUT   = X9,
              UNUSED
   )

%NODE( LOG
   PRIMITIVE = VOR_LOG
   PRIM_IN    = NFSS*NS/DASC,
              ASGN,
              2,
              0.0E0,
              X9   THRESHOLD = NFSS*NS/DASC
   PRIM_OUT   = X10
   )

%NODE( LRQT
   PRIMITIVE = DFC_LRQT
   PRIM_IN    = NFSS*NS/DASC,
              REQ(1),
              REQ(2),
              REQ(3),
              REQ(4),
              REQ(5),
              REQ(6),
              X10   THRESHOLD = NFSS*NS/DASC
   PRIM_OUT   = X11
   )

%NODE( HDI
   PRIMITIVE = DFC_HDI
   PRIM_IN    = NFSS*NS/DASC,
              1,
              1,
              HEADER,
              MNA,
              DM1,
              DM2,
              X11   THRESHOLD = NFSS*NS/DASC
   PRIM_OUT   = UNUSED,
              UNUSED,
              ASOT
   )
```

The Q003 Graph is converted to a Domain Primitive Graph. This requires the following substitutions/modifications:

1. For each node, replace the Q003 primitive with the corresponding Domain Primitive. The corresponding Domain Primitive can be determined by referring to the "Generalized Mapping Q003 Primitives to Domain Primitives." Change the `PRIM_IN` and `PRIM_OUT` lists as required, according to the information found in "Mapping of Parameters Q003 Primitives to Domain Primitives." Reference the "Domain Primitive Descriptions" and the Q003 Descriptions as necessary.

2. Some primitives may need to be changed due to how the primitive is being used. As will be shown in the example, the primitive `DFC_FCTR` is mapped first to `D_FLOC` based on the entry in the "Generalized Mapping Q003 Primitives to Domain Primitives."

Upon further examination of how the primitive is used (namely one output queue), the functionality required is that of D_FANIN.

Some variables may have to be created or modified to satisfy the requirements of the Domain Primitive. As an example, D_FANIN requires a variable (P) specifying a two dimensional of elements that are to be output onto each output queue, whereas DFC_FCTR requires only a single dimension array.

3. In some case, additional nodes must be added to the graph. As an example, the Domain Primitive D_LRQT, as currently implemented, does not convert FLOAT input to INT output. Therefore, it is necessary to explicitly convert the output queue by adding a node with a D_RTOI primitive and also adding a queue to connect the two nodes.

4. The AN/UYS-2 single precision modes are 16 bit entities and double precision modes are 32 bit entities. For the MCCI system, single precision is nominally 32 bit (target dependent). The modes of GIPs, VARs, and QUEUEs should be converted in that all double precision entities should be modified to single precision (e.g. DFLOAT => FLOAT.)

Performing these modifications on the ASNP Q003 graph yields the following Domain Primitive Graph:

```
%GRAPH( ASNP             %%  Domain Primitive Version
   GIP      =
             DASC   : INT,
             NAS    : INT,
             NS     : INT,
             FFTSZ  : INT,
             NIF    : INT,
             NFSS   : INT,
             BB     : INT,
             ASWIND : INT ARRAY(2),
             REQ    : FLOAT ARRAY(6),
             MNA    : INT ARRAY(2),
             DM1    : INT,
             DM2    : INT,
             %% V Array Size on output queue
             %% nominally KK = (NS*NFSS)/DASC + 8
             %%
             KK     : INT
   VAR      = ASNP_VALVE : INT,
             ASGN   : FLOAT,
             HEADER : INT ARRAY(8)
   INPUTQ   = MEF    : FLOAT,
             OMNI   : CFLOAT,
             CARD   : CFLOAT
   OUTPUTQ  = ASOT   : INT V_ARRAY(KK) )

%GIP( SPL1_BLS : INT ARRAY(2) INITIALIZE TO {FFTSZ, 1} )
%GIP( SPL2_BLS : INT ARRAY(2) INITIALIZE TO {NAS, 1} )
%GIP( P_FANIN  : INT ARRAY(4) INITIALIZE TO {NS*FFTSZ, 0, 0, NS*FFTSZ} )

%QUEUE( X1  : CFLOAT )
%QUEUE( X2  : CFLOAT )
```

```
%QUEUE( X3  : CFLOAT )
%QUEUE( X4  : CFLOAT )
%QUEUE( X5  : FLOAT )
%QUEUE( X6  : CFLOAT )
%QUEUE( X7  : CFLOAT )
%QUEUE( X8  : CFLOAT )
%QUEUE( X9  : FLOAT )
%QUEUE( X10 : FLOAT )
%QUEUE( X11 : FLOAT )
%QUEUE( X12 : INT )

%NODE( FCTR                 %%  Selects either OMNI or CARD input
                            %% based on BB
                            %%    BB = 1 => OMNI
                            %%    BB otherwise => CARD
   PRIMITIVE = D_FANIN      %% changed to FANIN from FLOC
   PRIM_IN   = NS*FFTSZ,
               2,
               P_FANIN,
               BB,
               FAMILY[OMNI, CARD] THRESHOLD = NS*FFTSZ
   PRIM_OUT  = X1,
               UNUSED
   )

%NODE( SPL1
   PRIMITIVE = D_SPL
   PIP_IN    = ASNP_VALVE
   PRIM_IN   = DASC*FFTSZ,
               1,
               SPL1_BLS,
               X1  THRESHOLD = NS*FFTSZ
   PRIM_OUT  = FAMILY[X2]  VARIABLE VALVE = ASNP_VALVE
   )

%NODE( SPL2
   PRIMITIVE = D_SPL
   PIP_IN    = ASNP_VALVE
   PRIM_IN   = DASC*NAS,
               1,
               SPL2_BLS,
               MEF THRESHOLD = NS*NAS
   PRIM_OUT  = FAMILY[X5]  VARIABLE VALVE = ASNP_VALVE
   )

%NODE( REORD
   PRIMITIVE = D_REORD
   PIP_IN    = ASNP_VALVE
   PRIM_IN   = FFTSZ,
               FFTSZ,
               1,
               (FFTSZ/2)+1,
               (FFTSZ/2)+2,
               FFTSZ,
               X2  THRESHOLD = (NS*FFTSZ)/DASC
   PRIM_OUT  = X3 )

%NODE( SPL3
   PRIMITIVE = D_SPL
   PRIM_IN   = FFTSZ,
```

```
                1,
                ASWIND,
                X3   THRESHOLD = (NS*FFTSZ)/DASC
   PRIM_OUT   = FAMILY[X4] )

%NODE( MUL
   PRIMITIVE = D_VMUL
   PRIM_IN    = (NS*NAS)/DASC,
                0,
                X5   THRESHOLD = (NS*NAS)/DASC,
                X4   THRESHOLD = (NS*NAS)/DASC
   PRIM_OUT   = X6 )

%NODE( REORD2
   PRIMITIVE = D_REORD
   PRIM_IN    = NAS,
                NIF,
                1,
                (NAS-1)/2,
                (NAS+1)/2,
                NAS,
                X6   THRESHOLD = (NS*NAS)/DASC
   PRIM_OUT   = X7 )

%NODE( FFT
   PRIMITIVE = D_FFT
   PRIM_IN    = NIF,
                NFSS,
                1,
                ((NIF-NFSS)/2)+1,
                UNUSED,
                X7   THRESHOLD = (NS*NIF)/DASC
   PRIM_OUT   = X8 )

%NODE( PWR
   PRIMITIVE = D_PWR
   PRIM_IN    = NFSS,
                UNUSED,
                X8   THRESHOLD = (NS*NFSS)/DASC
   PRIM_OUT   = X9,
                UNUSED )

%NODE( LOG
   PRIMITIVE = D_LOG
   PRIM_IN    = (NS*NFSS)/DASC,
                2,
                ASGN,
                0.0E0,
                X9   THRESHOLD = (NS*NFSS)/DASC
   PRIM_OUT   = X10 )

%NODE( LRQT
   PRIMITIVE = D_LRQT
   PRIM_IN    = (NS*NFSS)/DASC,
                REQ(1),
                REQ(2),
                REQ(5),
                REQ(6),
                REQ(3),
                REQ(4),
```

```
               X10 THRESHOLD = (NS*NFSS)/DASC
   PRIM_OUT  = X11 )

%NODE( CNVRT
   PRIMITIVE = D_RTOI
   PRIM_IN   = NS*NFSS/DASC,
               UNUSED,
               UNUSED,
               X11 THRESHOLD = NS*NFSS/DASC
   PRIM_OUT  = X12
   )

%NODE( HDI
   PRIMITIVE = D_HDI
   PRIM_IN   = (NS*NFSS)/DASC,
               1,
               HEADER,
               MNA,
               DM1,
               DM2,
               X12 THRESHOLD = (NS*NFSS)/DASC
   PRIM_OUT  = UNUSED,
               UNUSED,
               ASOT
   )
%ENDGRAPH
```

The next step is to examine how the chain is used in the application. In this example, the node referencing `CHN_ASNP` is in the graph `CWASCAN`. The `%NODE` statement extracted from `CWASCAN` is:

```
%NODE  ( ASCN
        PRIMITIVE = CHN_ASNP
        PIP_IN    = CARD,
                    ASEL,
                    CARDBEAR,
                    VLV
                       THRESHOLD = 1
        PRIM_IN   = SDNS,
                    NAS,
                    NS,
                    FFTSZ,
                    NIF,
                    NFSS,
                    BB(IF CARD+ASEL+CARDBEAR EQ 0 THEN 1 ELSE 2),
                    (IF VLV EQ MN THEN 1 ELSE 0),
                    ASWIND,
                    ASGN,
                    REQ,
                    HEADER,
                    MNA,
                    DM1
                       THRESHOLD = 1,
                    DM2,
                    MEF
                       THRESHOLD = NS*NAS,
                    OMNI
```

```
                    THRESHOLD = NS*FFTSZ,
                 CRD
                    THRESHOLD = NS*FFTSZ
    PRIM_OUT   = ASOT
    PIP_OUT    = DM1
                    PRODUCE = 1 OF 0,
                 VLV
                    VARIABLE PRODUCE = 1 OF
                                       (IF VLV EQ MN THEN 1 ELSE VLV+1)
)
```

From the information in the %NODE statement, the chain description, and the graph header from the ASNP Domain Primitive graph, the %SUBGRAPH statement that will be inserted into CWASCAN to replace the %NODE statement can be constructed.

First, the subgraph must be given a name and the underlying graph must be referenced:

```
%SUBGRAPH ( ASCN
            GRAPH    = ASNP
```

The INPUTQ and OUTPUTQ lists are readily extracted from the %NODE statement. The queues MEF, OMNI, and CRD are input queues as expected. The queue ASOT is an output queue as expected. The parameter DM1 is also a queue.

There is also a feedback queue (both an input and output from the same node) named VLV, associated with a PIP_IN and PIP_OUT. This will be ignored for the moment.

This leads to the following list.

```
INPUTQ   = DM1,
           MEF,
           OMNI,
           CRD
OUTPUTQ = DM1,
           ASOT
```

Next the GIP and VAR lists are constructed from the %NODE statement and the graph CWASCAN.

The following %NODE statement parameters are formal GIPs to the graph CWASCAN : SDNS, NS, FFTSZ, NIF, and NFSS.

The following %NODE statement parameters are local GIPs to the graph CWASCAN : NAS, REQ, BB, and MNA.

The following %NODE statement parameters are formal VARs to the graph CWASCAN : ASWIND, ASGN, HEADER.

The %NODE statement entry corresponding to the ASNP_VALVE parameter is an expression and therefore ASNP_VALVE must be a VAR.

Based on these observations, the following GIP and VAR lists are constructed:

```
GIP      = SDNS,
           NAS,
           NS,
           FFTSZ,
           NIF,
           NFSS,
           BB,
           REQ,
           MNA,
           DM2
VAR      = ASNP_VALVE ,
           ASWIND,
           ASGN,
           HEADER
```

Putting this together, the preliminary %SUBGRAPH statement becomes:

```
%SUBGRAPH ( ASCN
            GRAPH    = ASNP
            GIP      = SDNS,
                       NAS,
                       NS,
                       FFTSZ,
                       NIF,
                       NFSS,
                       BB,
                       REQ,
                       MNA,
                       DM2
            VAR      = ASNP_VALVE,
                       ASWIND,
                       ASGN,
                       HEADER
            INPUTQ   = VLV,
                       DM1,
                       MEF,
                       OMNI,
                       CRD
            OUTPUTQ  = VLV,
                       DM1,
                       ASOT
          )
```

The preliminary %SUBGRAPH statement must now be reconciled with the graph header for the ASNP Domain Primitive graph. It must be remembered that the %SUBGRAPH statement contains actual arguments while the ASNP Domain Primitive graph contains formal arguments. (Actual names may be the same as formal names but are not required to be identical.)

The following changes must be made to the ASNP Domain Primitive graph header:

ASWIND was declared as a GIP and needs to be a VAR.

DM1 was declared as a GIP and needs to be an INPUTQ and an OUTPUTQ.

The following changes must be made in the %SUBGRAPH statement:

The CWASCAN GIP parameter VASZ must be passed to the ASNP Domain Primitive graph parameter KK. This parameter is used to set the maximum size of the v_array output queue ASOT.

The ASNP Domain Primitive graph header becomes:

```
%GRAPH( ASNP               %%  Domain Primitive Version
   GIP      =
            DASC     : INT,
            NAS      : INT,
            NS       : INT,
            FFTSZ    : INT,
            NIF      : INT,
            NFSS     : INT,
            BB       : INT,
            REQ      : FLOAT ARRAY(6),
            MNA      : INT ARRAY(2),
            DM2      : INT,
            %% V Array Size on output queue
            %% nominally KK = (NS*NFSS)/DASC + 8
            %%
            KK       : INT
   VAR      = ASNP_VALVE : INT,
            ASWIND : INT ARRAY(2),
            ASGN     : FLOAT,
            HEADER : INT ARRAY(8)
   INPUTQ   = DM1      : INT,
            MEF      : FLOAT,
            OMNI     : CFLOAT,
            CARD     : CFLOAT
   OUTPUTQ = DM1      : INT,
            ASOT     : INT V_ARRAY(KK) )
```

The %SUBGRAPH statement becomes:

```
%SUBGRAPH ( ASCN
            GRAPH    = ASNP
            GIP      = SDNS,
                       NAS,
                       NS,
                       FFTSZ,
                       NIF,
                       NFSS,
                       BB,
                       REQ,
                       MNA,
                       DM2,
                       VASZ
            VAR      = ASNP_VALVE,
                       ASWIND,
                       ASGN,
                       HEADER
```

```
        INPUTQ   = DM1,
                   MEF,
                   OMNI,
                   CRD
        OUTPUTQ = DM1,
                   ASOT
    )
```

Finally, the remaining items in the CWASCAN graph %NODE statement (PIP_IN, PIP_OUT, and expressions) must be included into the application specific ASNP Domain Primitive graph.

First, the expression associated with BB is considered. The expression contains three variables (CARD, ASEL, and CARDBEAR) that are formal VARs to the CWASCAN graph. These variables must be formals to the ASNP Domain Primitive graph. The VAR list becomes:

```
        VAR      = CARD,
                   ASEL,
                   CARD_BEAR,
                   ASNP_VALVE,
                   ASWIND,
                   ASGN,
                   HEADER
```

Note that the order is arbitrary provided that the calling %SUBGRAPH statement and the graph header mate correctly.

The expression is evaluated to select an element from the variable BB that is a local GIP to the CWASCAN graph. The variable BB contains two elements {1, 2}. It is interesting to note that the expression evaluates to these same values. Because of this, it is possible to eliminate the variable BB, and just use the expression. It was decided to just use the expression and eliminate BB to avoid the run-time slicing of BB.

Next, the functionality of the feedback queue VLV is considered. This functions as a modulo counter. Every time the node executes, the value of the integer token placed onto the feedback queue is increased by one until the value at the beginning of execution of the node is equal to MN. When that occurs, a value of 1 is produced. This functionality must be placed into the ASNP Domain Primitive subgraph via PIP_IN and PIP_OUT mechanism associated with the FCTR node. The variable MN which is part of the valve expression must be added to the graph header. This variable is a formal GIP to the CWASCAN graph and will therefore be a formal GIP to the ASNP Domain Primitive Graph.

```
%NODE( FCTR
   PRIMITIVE = D_FANIN
   PIP_IN    = VLV
   PRIM_IN   = NS*FFTSZ,
               2,
               P_FANIN,
               BB,
               FAMILY[OMNI, CARD] THRESHOLD = NS*FFTSZ
```

```
   PRIM_OUT   = X1,
                UNUSED
   PIP_OUT    = VLV  VARIABLE PRODUCE = 1 OF (IF VLV EQ MN THEN 1 ELSE VLV+1)
   )
```

The queue VLV can either be declared as a local queue or as both a formal INPUTQ and a formal OUTPUTQ. If it is declared as a local queue, the initialization contained in the CWASCAN graph must be included in the ASNP Domain Primitive graph. It was decided to make it formal to maintain consistency with the original graph.

Additionally, since the value of VLV is used by two other nodes (SPL1 and SPL2), the value of VLV must be passed to these two nodes. This is done by creating two queues of mode INT, creating two PIP_OUTs on the FCTR node one for each queue, and creating a PIP_IN on the SPL1 and SPL2 nodes.

Next the expression ((IF VLV EQ MN THEN 1 ELSE 0)) in the PRIM_IN list that is associated with the parameter ASNP_VALVE is considered. This variable is used by two nodes (SPL1 and SPL2). The value for VLV was declared as a PIP_IN in the modifications described in the previous paragraph. The expression is substituted for the parameter ASNP_VALVE in the PRIM_IN list for nodes SPL1 and SPL2. The parameter ASNP_VALVE is no longer used and is removed from the formal VAR list and the %SUBGRAPH statement.

When these modifications have been included, the ASNP Domain Primitive graph becomes:

```
%%  Domain Primitive Version
%%
%GRAPH( ASNP
   GIP      = DASC : INT,
              NAS : INT,
              NS : INT,
              FFTSZ : INT,
              NIF : INT,
              NFSS : INT,
              BB : INT ARRAY(2),
              REQ : FLOAT ARRAY(6),
              MNA : INT ARRAY(2),
              DM2 : INT,

              %% V Array Size on output queue
              %% nominally KK = (NS*NFSS)/DASC + 8
              %%
              KK : INT,
              MN : INT
   VAR      = CARDI : INT,
              ASEL : INT,
              CARD_BEAR : INT,
              ASWIND : INT ARRAY(2),
              ASGN : FLOAT,
              HEADER : INT ARRAY(8)
   INPUTQ   = VLV : INT,
              DM1 : INT,
              MEF : FLOAT,
```

```
             OMNI : CFLOAT,
             CARD : CFLOAT
   OUTPUTQ = VLVP : INT,
             DM1P : INT,
             ASOT : INT V_ARRAY(KK) )
%GIP( SPL1_BLS : INT ARRAY(2) INITIALIZE TO {FFTSZ, 1} )
%GIP( SPL2_BLS : INT ARRAY(2) INITIALIZE TO {NAS, 1} )
%GIP( P_FANIN : INT ARRAY(4) INITIALIZE TO {NS*FFTSZ, 0, 0,
    NS*FFTSZ} )
%QUEUE( X1 : CFLOAT )
%QUEUE( X2 : CFLOAT )
%QUEUE( X3 : CFLOAT )
%QUEUE( X4 : CFLOAT V_ARRAY ((NS*NAS)/DASC))
%QUEUE( X5 : FLOAT V_ARRAY ((NS*NAS)/DASC))
%QUEUE( X6 : CFLOAT )
%QUEUE( X7 : CFLOAT )
%QUEUE( X8 : CFLOAT )
%QUEUE( X9 : FLOAT )
%QUEUE( X10 : FLOAT )
%QUEUE( X11 : FLOAT )
%QUEUE( X12 : INT )
%QUEUE( VLV1 : INT )
%QUEUE( VLV2 : INT )

%%  Selects either OMNI or CARD input
%% based on BB
%%    BB = 1 => OMNI
%%    BB otherwise => CARD
%% changed to FANIN from FLOC
%%
%NODE( FCTR
   PRIMITIVE = D_FANIN
   PIP_IN    = CARDI,
               ASEL,
               CARD_BEAR,
               VLV THRESHOLD = 1
   PRIM_IN   = NS*FFTSZ,
               2,
               P_FANIN,
               (IF ((CARDI+ASEL)+CARD_BEAR) EQ 0 THEN 1 ELSE 2),
               FAMILY[OMNI,CARD] THRESHOLD = NS*FFTSZ
   PRIM_OUT  = X1,
               UNUSED
   PIP_OUT   = VLVP VARIABLE PRODUCE = 1 OF (IF VLV EQ MN THEN 1 ELSE VLV+1),
               VLV1 VARIABLE PRODUCE = 1 OF VLV,
               VLV2 VARIABLE PRODUCE = 1 OF VLV )
%NODE( SPL1
   PRIMITIVE = D_SPL
   PIP_IN    = VLV1 THRESHOLD = 1
   PRIM_IN   = DASC*FFTSZ,
               1,
               SPL1_BLS,
               X1  THRESHOLD = NS*FFTSZ
   PRIM_OUT  = FAMILY[X2] VARIABLE VALVE = (IF VLV1 EQ MN THEN 1 ELSE 0) )
%NODE( SPL2
   PRIMITIVE = D_SPL
   PIP_IN    = VLV2 THRESHOLD = 1
   PRIM_IN   = DASC*NAS,
               1,
               SPL2_BLS,
```

```
                 MEF THRESHOLD = NS*NAS
   PRIM_OUT    = FAMILY[X5] VARIABLE VALVE = (IF VLV2 EQ MN THEN 1 ELSE 0) )
%NODE( REORD
   PRIMITIVE = D_REORD
   PRIM_IN     = FFTSZ,
                 FFTSZ,
                 1,
                 (FFTSZ/2)+1,
                 (FFTSZ/2)+2,
                 FFTSZ,
                 X2  THRESHOLD = (NS*FFTSZ)/DASC
   PRIM_OUT    = X3 )
%NODE( SPL3
   PRIMITIVE = D_SPL
   PRIM_IN     = FFTSZ,
                 1,
                 ASWIND,
                 X3  THRESHOLD = (NS*FFTSZ)/DASC
   PRIM_OUT    = FAMILY[X4] )
%NODE( MUL
   PRIMITIVE = D_VMUL
   PRIM_IN     = (NS*NAS)/DASC,
                 0,
                 X5  THRESHOLD = (NS*NAS)/DASC,
                 X4  THRESHOLD = (NS*NAS)/DASC
   PRIM_OUT    = X6 )
%NODE( REORD2
   PRIMITIVE = D_REORD
   PRIM_IN     = NAS,
                 NIF,
                 1,
                 (NAS-1)/2,
                 (NAS+1)/2,
                 NAS,
                 X6  THRESHOLD = (NS*NAS)/DASC
   PRIM_OUT    = X7 )
%NODE( FFT
   PRIMITIVE = D_FFT
   PRIM_IN     = NIF,
                 NFSS,
                 1,
                 ((NIF-NFSS)/2)+1,
                 UNUSED,
                 X7  THRESHOLD = (NS*NIF)/DASC
   PRIM_OUT    = X8 )
%NODE( PWR
   PRIMITIVE = D_PWR
   PRIM_IN     = NFSS,
                 UNUSED,
                 X8  THRESHOLD = (NS*NFSS)/DASC
   PRIM_OUT    = X9,
                 UNUSED )
%NODE( LOG
   PRIMITIVE = D_LOG
   PRIM_IN     = (NS*NFSS)/DASC,
                 2,
                 ASGN,
                 0.0E0,
                 X9  THRESHOLD = (NS*NFSS)/DASC
   PRIM_OUT    = X10 )
```

```
%NODE( LRQT
   PRIMITIVE = D_LRQT
   PRIM_IN   = (NS*NFSS)/DASC,
               REQ(1),
               REQ(2),
               REQ(5),
               REQ(6),
               REQ(3),
               REQ(4),
               X10 THRESHOLD = (NS*NFSS)/DASC
   PRIM_OUT  = X11 )
%NODE( CNVRT
   PRIMITIVE = D_RTOI
   PRIM_IN   = (NS*NFSS)/DASC,
               UNUSED,
               UNUSED,
               X11 THRESHOLD = (NS*NFSS)/DASC
   PRIM_OUT  = X12 )
%NODE( HDI
   PRIMITIVE = D_HDI
   PRIM_IN   = (NS*NFSS)/DASC,
               1,
               HEADER,
               MNA,
               DM1 THRESHOLD = 1,
               DM2,
               X12 THRESHOLD = (NS*NFSS)/DASC
   PRIM_OUT  = UNUSED,
               UNUSED,
               ASOT
   PIP_OUT   = DM1P PRODUCE = 1 OF 0 )
%ENDGRAPH
```

The modified `CWASCAN` graph is obtained by replacing the `%NODE` statement by the `%SUBGRAPH` statement and making the other changes discussed. The modified `CWASCAN` graph is:

```
%%
%% ../src/dccwas.grf:
%%
%%***********************************************************%%
%%                                                           %%
%%                 GRAPH 'CWASCAN        '                   %%
%%                 SPGN generated from GRED                  %%
%%                 Thu Mar 18 13:23:41 1993                  %%
%%                                                           %%
%%***********************************************************%%
%%
%%     Unit: DICASS CWL,M,S A-Scan (150208, 150308, 150408)
%%
%%     Designed by: D. C. Lui
%%     Coded by: D. C. Lui
%%     Tested by: D. C. Lui and V. J. Izzo
%%
%%     Purpose: Performs omni/cardioid selection, windowing, reorder, zero
%%              fill, IFFT, detection, log and scaling, requantization and
```

```
%%              AIU header insert.
%%
%%      Initial Conditions: See the GIP list and the QUEUE list below for
%%                          initial conditions.
%%
%%      Inputs: Cardioid and omni FFT data from Input Process Unit, normalized
%%              window weights from Cardioid Mean Estimation Unit and control
%%              parameters.
%%
%%      Outputs: A-Scan time series data for AIU via IOP.
%%
%%      Requirements: SRS Section 3.4.2.16.2.2.1.17 to 3.4.2.16.2.2.1.21,
%%                    02/01/90
%%

%GRAPH ( CWASCAN
    GIP     = VASZ      : INT,   %% VASZ => Max V_ARRAY size
              PB        : INT,   %% PB => Number of passband bins
              PTYPE     : INT,   %% PTYPE => Ping type: CWL=1, CWM=2, CWS=3
              DASC      : INT,   %% DASC => Input decimation rate CWL,M=10 CWS=4
              FFTSZ     : INT,   %% FFTSZ => FFT size
                    %% NFSS => Number of bins selected for output from IFFT
                    %% NFSS =>   CWL,M=205, CWS=20
              NFSS      : INT,
              NIF       : INT,   %% NIF => IFFT size
              NS        : INT,   %% NS => Number of scans per processing block.
                    %% MN => Number of input blocks CWL,CWM=DASC/NS, CWS=1
              MN        : INT,
                    %% SDNS => Number of scans for input decimation.
                    %% SDNS =>  CWL,CWM=NS CWS=DASC
              SDNS      : INT
    VAR     =
                    %% CARD  => Normal/Cardioid selection 0=normal, 1=cardioid
              CARD      : INT,
              ASEL      : INT,   %% ASEL => Audio selection 0=omni, 1=cardioid
              CARDBEAR : INT,    %% CARDBEAR => Bearing enhance
                                 %% CARDBEAR => 0=omni, 1=cardiod
              ASGN   :  DFLOAT, %% ASGN => A-Scan amplitude adjustment factor
              HEADER    : INT ARRAY(8), %% HEADER => AIU header
              ASWIND    : INT ARRAY(2)  %% ASWIND => band selection array for
                                        %%           DCP_SPL
    INPUTQ  = OMNI      : DCFLOAT,  %% Omni data from Input Process Unit
              CRD       : DCFLOAT,  %% Cardioid data from Input Process Unit
                    %% Normalized weights from Cardioid Mean Estimation Unit
              MEF       : DFLOAT
    OUTPUTQ =
                    %% A-Scan output to Flow Control LLCSC
              ASOT      : INT V_ARRAY(VASZ)
    )

          %% DECLARATIONS section (%GIP, %VAR, %QUEUE)

%GIP ( AFLL : DFLOAT   %% 2log10, base 2 for lower clipping in requantization
              INITIALIZE TO 6.643856188E00 )

%GIP ( AFC : DFLOAT    %% Requantization conversion factor
             INITIALIZE TO 1.28E+02/AFLL )

%GIP ( AFLU : DFLOAT   %% Upper clipping for requantization
              INITIALIZE TO (1.27E+02/1.28E+02)*AFLL )
```

```
%GIP ( AFSA : DFLOAT   %% Requantization offset
              INITIALIZE TO 0.0E+00 )

%GIP ( REQ : DFLOAT ARRAY(6)   %% Requantization array
             INITIALIZE TO { AFC, AFSA*AFC, AFLU-AFSA, (-1.0E+00*AFLL)-AFSA,
                             (1.27E+02/AFC)-AFSA, (-1.27E+02/AFC)-AFSA } )

%GIP ( ASCANFG : INT   %% A-Scan data type for AIU header
                 INITIALIZE TO 24 )

%GIP ( DM2 : INT       %% Data mask 2 for AIU header
             INITIALIZE TO PTYPE*512+ASCANFG*4 )

%GIP ( MNA : INT ARRAY(2)      %% Words to be ORed in AIU header
             INITIALIZE TO { 6, 6 } )

%GIP ( MNS : INT       %% Total number of scans
             INITIALIZE TO MN*NS )

%GIP ( NID : INT       %% Processing block size after bin reorder and padding.
             INITIALIZE TO (MNS/DASC)*NIF )

%GIP ( NSD : INT       %% Processing block size after IFFT
             INITIALIZE TO (MNS/DASC)*NFSS )

%GIP ( NAS : INT       %% Number of FFT bins selected for a a-scan band
             INITIALIZE TO PB+6 )

%GIP ( NAD : INT       %% Processing block size after FFT bin selection
             INITIALIZE TO (MNS/DASC)*NAS )

%GIP ( OUTST : INT     %% Output starting bin number for IFFT output
               INITIALIZE TO ((NIF-NFSS)/2)+1 )

%% Omni/Cardioid selection array for input flow control
%GIP ( BB : INT ARRAY(2) INITIALIZE TO { 1, 2 } )

%QUEUE ( DM1 : INT     %% First data block flag
               INITIALIZE TO 1 OF 8192 )

%QUEUE ( VLV : INT     %% Valve control
               INITIALIZE TO 1 OF MN )

                %% TOPOLOGY section (%NODE, %SUBGRAPH)

%%NODE ( ASCN
%%        PRIMITIVE = CHN_ASNP
%%        PIP_IN    = CARD,
%%                    ASEL,
%%                    CARDBEAR,
%%                    VLV
%%                       THRESHOLD = 1
%%        PRIM_IN   = SDNS,
%%                    NAS,
%%                    NS,
%%                    FFTSZ,
%%                    NIF,
%%                    NFSS,
%%                    BB(IF CARD+ASEL+CARDBEAR EQ 0 THEN 1 ELSE 2),
```

```
%%                      (IF VLV EQ MN THEN 1 ELSE 0),
%%                      ASWIND,
%%                      ASGN,
%%                      REQ,
%%                      HEADER,
%%                      MNA,
%%                      DM1
%%                         THRESHOLD = 1,
%%                      DM2,
%%                      MEF
%%                         THRESHOLD = NS*NAS,
%%                      OMNI
%%                         THRESHOLD = NS*FFTSZ,
%%                      CRD
%%                         THRESHOLD = NS*FFTSZ
%%          PRIM_OUT  = ASOT
%%          PIP_OUT   = DM1
%%                         PRODUCE = 1 OF 0,
%%                      VLV
%%                         VARIABLE PRODUCE = 1 OF
%%                                            (IF VLV EQ MN THEN 1 ELSE VLV+1)
%%        )

%SUBGRAPH ( ASCN
            GRAPH     = ASNP
            GIP       = SDNS,
                        NAS,
                        NS,
                        FFTSZ,
                        NIF,
                        NFSS,
                        BB,
                        REQ,
                        MNA,
                        DM2,
                        VASZ,
                        MN
            VAR       = CARD,
                        ASEL,
                        CARDBEAR,
                        ASWIND,
                        ASGN,
                        HEADER
            INPUTQ    = VLV,
                        DM1,
                        MEF,
                        OMNI,
                        CRD
            OUTPUTQ = VLV,
                        DM1,
                        ASOT
          )

%ENDGRAPH
```

6.4 DICASS Conversion

A version of the DICASS Sonobuoy application was converted from the AN/UYS-2 implementation to a DPAG implementation. This process involved:

a. Converting the AN/UYS-2 DICASS graph and referenced subgraphs from nodes referencing Q003 primitives to nodes referencing Domain Primitives as described in Section 6.2 Converting AN/UYS-2 Graphs.

b. Converting the ASNP and BDWF chains from AN/UYS-2 chains to Domain Primitive Subgraphs as described in Section 6.3 AN/UYS-2 Chains.

c. Implementing eighteen new Domain Primitives that implement sonobuoy, display formatting, and/or DICASS specific processing.

6.4.1 Graph and Subgraphs

The philosophy regarding the conversion of the DICASS graph and related subgraphs was to make as few modifications as possible, especially in regards to the DICASS graph formal interface. This philosophy may or may not be desirable depending upon the target platform on which the ported application will execute. Most AN/UYS-2 applications contain a large amount of display formatting. It should be remembered that the AN/UYS-2 Arithmetic Processors are 16 bit machines, and some of the formatting is based on this. A natural question arises when porting to a 32 bit (or 64 bit) machine. How should the data be packed? If the display is being replaced, the formatting will likely change and this should be considered as part of the conversion. Additionally, some AN/UYS-2 applications such as DICASS store data in the ISC memory for reprocessing. If the ISC is not being used in the new target platform, this data will have to be stored in a different place, possibly in memory located on the target platform. Finally, since the Command Program is not readily reusable, should the data from the Command Program be kept in the same format?

DICASS output is passed to tracking processing. The tracking processing, including the AIU graph which distributes the feedback parameters, was not converted.

In addition to the modifications discussed in Section 6.2 Converting AN/UYS-2 Graphs, the following modifications were made:

1. Converted all families of Graph Instantiation Parameters (GIPs) into GIP arrays. In all cases, the members of the family were single integers, therefore the conversion was straightforward.

Example:

Declaration:
```
%GIP( [1..2]ZTHR : INT
        INITIALIZE [1]ZTHR TO RYA
        INITIALIZE [2}ZTHR TO NPADN )
```
becomes:
```
%GIP( ZTHR : INT ARRAY(2) INITIALZE TO {RYA, NPADN} )
```
Usage:
```
FAMILY[I=PADQ,ZPAD] THRESHOLD = [I]ZTHR
```
becomes:
```
FAMILY[I=PADQ,ZPAD] THRESHOLD = ZTHR(I)
```

These types of conversion were necessary because the MCCI Autocoding Toolset does not currently support GIP families.

2. Modified the graph as necessary to eliminate multi-element slicing of entities (GIPs and VARs). This entailed changing the basic entity from a GIP or VAR to a queue and specifying READ and OFFSET amounts.

```
%VAR( NOTCHW : FLOAT ARRAY(2,NFT) INITIALIZE TO {NFT OF 1.0E+00,
      ((NFT/2)-SBB)-3 OF 1.0E+00, 0.701201E+00, 0.242273E+00,
      0.23472E-01, (2*SBB)+1 OF 0.0E+00, 0.23472E-01, 0.242273E+00,
      0.701201E+00, ((NFT/2)-SBB)-4 OF 1.0E+00} )

%QUEUE( NOTCHW : FLOAT INITIALIZE TO NFT OF 1.0E+00,
      ((NFT/2)-SBB)-3 OF 1.0E+00, 0.701201E+00, 0.242273E+00,
      0.23472E-01, (2*SBB)+1 OF 0.0E+00, 0.23472E-01, 0.242273E+00,
      0.701201E+00, ((NFT/2)-SBB)-4 OF 1.0E+00 )

%NODE(SLICE1
      PRIMITIVE = D_REP
      PIP_IN    = AREVERB,
                  SLICE_TRIG THRESHOLD = 1
      PRIM_IN   = NFT,
                  1,
                  NOTCHW THRESHOLD = 2*NFT
                      READ = NFT
                      VARIABLE OFFSET = (AREVERB-1)*NFT
                      CONSUME = 0
      PRIM_OUT  = FAMILY[NOTCHW_VAR]
      PIP_OUT   = NOTCH_TRIG PRODUCE = 1 OF 1
)
```

For those few cases where the entity was used by more than one node, an additional node was inserted into the graph that performed an offset read from the queue and placed the data into a VAR.

```
%NODE( AWQ2VAR
      PRIMITIVE = D_REP
      PRIM_IN   = UNUSED,
      1,
      AW
            THRESHOLD = (4*19)*5
            READ = 19*5
            VARIABLE OFFSET = ((SONF-1)*19)*5
            CONSUME = 0
      PRIM_OUT  = FAMILY[AW_VAR]
    )
```

In order to execute these nodes only at graph start and graph re-initialization, a trigger queue input to the node as a PIP_IN was used to control when the node(s) was ready for execution. Further, one or more trigger queues were incorporated as outputs from the node (as a PIP_OUT) and input to the node requiring the VAR to ensure that the VAR was initialized correctly prior to use by another node.

This type of conversion was necessary because the MCCI Autocoding Toolset and data flow graph executing SRTS do not currently support slicing.

There were some minor edits where the AN/UYS-2 graph was passing an array or a single element of the array as a parameter to a node. These edits were of the form:

FOO(1..2, 1..N)	=>	FOO	%% entire array being passed.
FOO(1..1)	=>	FOO or FOO(1)	%% either entire array or single element passed.
BB(expr)	=>	expr	%% BB contained values equal to index.

6.4.2 Domain Primitives

The Domain Primitives that were implemented for DICASS are:

DCP_CGA - Channel Gain Adjust
DCP_CLS - DICASS Clustering
DCP_CRB - Center Reverberation Bin Estimation
DCP_INTD - Interpolation - Decimation
DCP_LAGI - Weighted Lag Integration
DCP_RINT - Running Integration
DFC_HDI - Header Insert
DFC_MCS - Mode Change Synchronization
DFC_PACK - Data Bit Pack
DFC_REQ - Requantization
DFC_VSCT - V_Array Selective Concatentation
DGP_HFMG - Hyperbolic FM Generation
SSP_AGC - Automatic Gain Control
SSP_CARD - Cardioid Formation
SSP_DCD - DIFAR Coherent Detection
SSP_SYNO - Synthetic Omni and Bearing Formation
SSP_ZDT - Zero Detection
VCM_DTH - DICASS Thresholding

The following Q003 Primitives are used in DICASS but were not implemented.

DMC_FXFL - Fixed to Float conversion. Data from ISC is FIXED with scale of 0. This is same as integer. Used integer to float conversion.
DFC_VPACK - Used VSCT instead.

Additionally, a (preliminary) version of MERGE was implemented.

Some of the Domain Primitives that were implemented are complex due to the generalized nature of the primitive. Mode Change Synchronization is an example. This primitive is used to ensure that graph variables updated by the Command Program occur at essentially the same time. The primitive permits different sizes of families for each of the five outputs and any output can be unused. In hindsight, some

of these primitives should have been implemented in a simpler fashion without being concerned with completely implementing all of the generalized functionality of the Q003 version.

6.4.3 Chains

The ASNP chain was converted into a subgraph using the conversion procedure described in Section 6.3 AN/UYS-2 Chains.

The BDWF chain was converted into five subgraphs using the conversion procedure described in Section 6.3 AN/UYS-2 Chains. Each mode of operation was implemented as a separate subgraph, and each referenced a common subgraph which implemented the tail-end processing.

6.4.4 Partitioning

The partitioning scheme implemented was based solely on partitioning requirements of the Autocoding Toolset. The restrictions imposed by the Toolset are that variable reads, variable consumes, and variable writes can only occur on queues that cross partition boundaries. "Variable" valves can only occur on partition output queues; however, by changing a "variable" valve to a valve and including the parameter values in the Graph Value Set, this can be encapsulated inside a partition at the expense of increased code size. The Merge construct must be in a partition by itself.

Based on these rules, certain Domain Primitives essentially force partition boundaries. However, in many cases, the complete flexibility of this type of Domain Primitive is not required. By using a Domain Primitive that performs the desired operation but does not have the flexibility, a partition boundary requirement may be eliminated. An example is using D_CAT instead of D_FANIN. D_FANIN permits run-time variation of how the data is concatenated, but in this mode requires that the output queue be an output from the partition. D_CAT also concatenates data, but without the run-time variation. In this demonstration, no effort was made to eliminate partition boundaries. This may or may not be possible for the DICASS application.

The partitioning performed for this demonstration resulted in 142 partitions.

The iconic form of one partition, P_CWSIN_4, is shown in Figure 10.

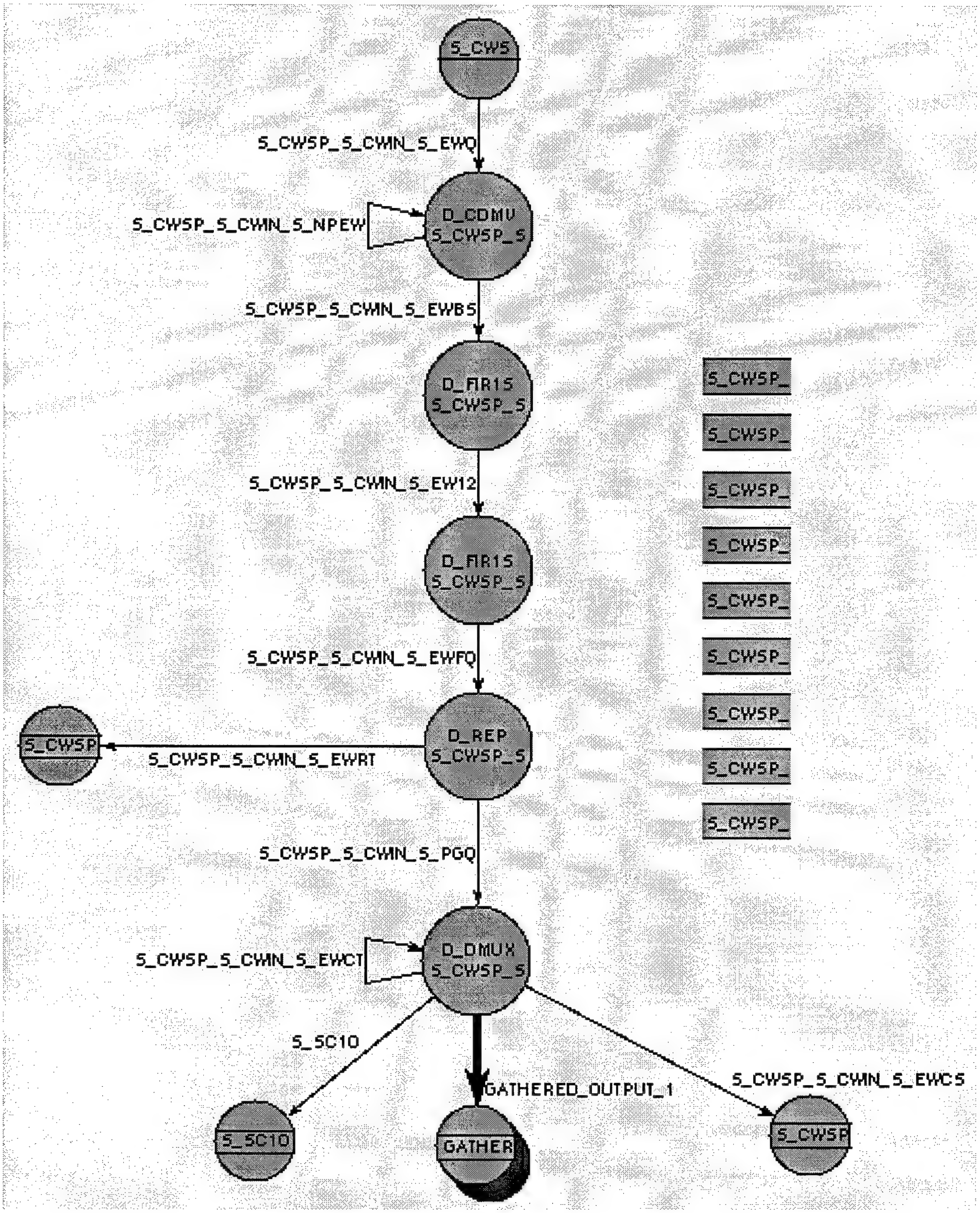


Figure 10. Partition P_CWSIN_4

The notational form of the same partition is:

```
%GRAPH (P_CWSIN_4
   INPUTQ = S_CWSP_S_CWIN_S_EWQ : FLOAT
   OUTPUTQ = S_SC10 : INT,
      S_CWSP_S_CWIN_S_EWCS : INT,
      S_CWSP_S_CWIN_S_EWRT : CFLOAT,
      [1..3]GATHERED_OUTPUT_1 : CFLOAT)

%GIP (S_CWSP_S_NF : INT
   INITIALIZE S_CWSP_S_NF TO 20)
%GIP (S_CWSP_S_CWIN_S_F : FLOAT ARRAY(1)
   INITIALIZE S_CWSP_S_CWIN_S_F TO {2.44140600000000E-01})
%GIP (S_CWSP_S_CWIN_S_FIRSZ1 : INT
   INITIALIZE S_CWSP_S_CWIN_S_FIRSZ1 TO 11)
%GIP (S_CWSP_S_CWIN_S_FIR1 : FLOAT ARRAY(S_CWSP_S_CWIN_S_FIRSZ1)
   INITIALIZE S_CWSP_S_CWIN_S_FIR1 TO {9.87871740000000E-03,
   -4.20027990000000E-04, -5.85211180000000E-02, 1.07341250000000E-03,
   2.98794630000000E-01, 4.98531370000000E-01, 2.98794630000000E-01,
   1.07341250000000E-03, -5.85211180000000E-02, -4.20027990000000E-04,
   9.87871740000000E-03})
%GIP (S_CWSP_S_CWIN_S_FIRSZ2 : INT
   INITIALIZE S_CWSP_S_CWIN_S_FIRSZ2 TO 39)
%GIP (S_CWSP_S_CWIN_S_FIR2 : FLOAT ARRAY(S_CWSP_S_CWIN_S_FIRSZ2)
   INITIALIZE S_CWSP_S_CWIN_S_FIR2 TO {-1.23575920000000E-03,
   8.70276560000000E-05, 2.53144400000000E-03, -7.23021220000000E-05,
   -4.79434850000000E-03, 1.93636250000000E-04, 8.44822920000000E-03,
   -2.12764510000000E-04, -1.37729710000000E-02, 3.47536320000000E-04,
   2.18234450000000E-02, -3.78957080000000E-04, -3.40838430000000E-02,
   5.03258610000000E-04, 5.51007170000000E-02, -5.15613120000000E-04,
   -1.00731690000000E-01, 6.01775770000000E-04, 3.16516400000000E-01,
   4.99431310000000E-01, 3.16516400000000E-01, 6.01775770000000E-04,
   -1.00731690000000E-01, -5.15613120000000E-04, 5.51007170000000E-02,
   5.03258610000000E-04, -3.40838430000000E-02, -3.78957080000000E-04,
   2.18234450000000E-02, 3.47536320000000E-04, -1.37729710000000E-02,
   -2.12764510000000E-04, 8.44822920000000E-03, 1.93636250000000E-04,
   -4.79434850000000E-03, -7.23021220000000E-05, 2.53144400000000E-03,
   8.70276560000000E-05, -1.23575920000000E-03})
%GIP (S_CWSP_S_CWIN_S_FS2 : FLOAT
   INITIALIZE S_CWSP_S_CWIN_S_FS2 TO 1.00000000000000E+00)
%GIP (S_CWSP_S_CWIN_S_NX : INT
   INITIALIZE S_CWSP_S_CWIN_S_NX TO (S_CWSP_S_NF * S_CWSP_S_NF))
%GIP (S_CWSP_S_CWIN_S_ISZ : INT
   INITIALIZE S_CWSP_S_CWIN_S_ISZ TO (4 * S_CWSP_S_CWIN_S_NX))
%QUEUE (S_CWSP_S_CWIN_S_EW12 : CFLOAT
   INITIALIZE S_CWSP_S_CWIN_S_EW12 TO ((S_CWSP_S_CWIN_S_FIRSZ2 - 2) * 3) OF <
   0.00000000000000E+00, 0.00000000000000E+00>)
%QUEUE (S_CWSP_S_CWIN_S_EWBS : CFLOAT
   INITIALIZE S_CWSP_S_CWIN_S_EWBS TO ((S_CWSP_S_CWIN_S_FIRSZ1 - 2) * 3) OF <
   0.00000000000000E+00, 0.00000000000000E+00>)
%QUEUE (S_CWSP_S_CWIN_S_EWCT : INT
   INITIALIZE S_CWSP_S_CWIN_S_EWCT TO 1 OF 0)
%QUEUE (S_CWSP_S_CWIN_S_EWFQ : CFLOAT)
%QUEUE (S_CWSP_S_CWIN_S_NPEW : INT
   INITIALIZE S_CWSP_S_CWIN_S_NPEW TO 1 OF 0)
%QUEUE (S_CWSP_S_CWIN_S_PGQ : CFLOAT)

%NODE (S_CWSP_S_CWIN_S_BSHF
```

```
   PRIMITIVE = D_CDMV
   PRIM_IN =
      S_CWSP_S_CWIN_S_ISZ,
      3,
      UNUSED,
      1024,
      S_CWSP_S_CWIN_S_F,
      S_CWSP_S_CWIN_S_FS2,
      S_CWSP_S_CWIN_S_NPEW
         THRESHOLD = 1,
      S_CWSP_S_CWIN_S_EWQ
         THRESHOLD = (S_CWSP_S_CWIN_S_ISZ * 3)
   PRIM_OUT =
      S_CWSP_S_CWIN_S_EWBS,
      S_CWSP_S_CWIN_S_NPEW)
%NODE (S_CWSP_S_CWIN_S_FD1
   PRIMITIVE = D_FIR1S
   PRIM_IN =
      ((S_CWSP_S_CWIN_S_ISZ + S_CWSP_S_CWIN_S_FIRSZ1) - 2),
      3,
      S_CWSP_S_CWIN_S_FIRSZ1,
      2,
      S_CWSP_S_CWIN_S_FIR1,
      S_CWSP_S_CWIN_S_EWBS
         THRESHOLD = (((S_CWSP_S_CWIN_S_ISZ + S_CWSP_S_CWIN_S_FIRSZ1) - 2) *3)
         READ = (((S_CWSP_S_CWIN_S_ISZ + S_CWSP_S_CWIN_S_FIRSZ1) - 2) * 3)
         CONSUME = (S_CWSP_S_CWIN_S_ISZ * 3)
   PRIM_OUT = S_CWSP_S_CWIN_S_EW12)
%NODE (S_CWSP_S_CWIN_S_FD2
   PRIMITIVE = D_FIR1S
   PRIM_IN =
      (((S_CWSP_S_CWIN_S_ISZ / 2) + S_CWSP_S_CWIN_S_FIRSZ2) - 2),
      3,
      S_CWSP_S_CWIN_S_FIRSZ2,
      2,
      S_CWSP_S_CWIN_S_FIR2,
      S_CWSP_S_CWIN_S_EW12
         THRESHOLD = ((((S_CWSP_S_CWIN_S_ISZ / 2) + S_CWSP_S_CWIN_S_FIRSZ2) -
         2) * 3)
         READ = ((((S_CWSP_S_CWIN_S_ISZ / 2) + S_CWSP_S_CWIN_S_FIRSZ2) - 2) *3)
         CONSUME = ((S_CWSP_S_CWIN_S_ISZ / 2) * 3)
   PRIM_OUT = S_CWSP_S_CWIN_S_EWFQ)
%NODE (S_CWSP_S_CWIN_S_EWR
   PRIMITIVE = D_REP
   PRIM_IN =
      (S_CWSP_S_CWIN_S_NX * 3),
      2,
      S_CWSP_S_CWIN_S_EWFQ
         THRESHOLD = (S_CWSP_S_CWIN_S_NX * 3)
   PRIM_OUT = FAMILY [S_CWSP_S_CWIN_S_EWRT, S_CWSP_S_CWIN_S_PGQ])
%NODE (S_CWSP_S_CWIN_S_DMXP
   PRIMITIVE = D_DMUX
   PIP_IN = S_CWSP_S_CWIN_S_EWCT
      THRESHOLD = 1
   PRIM_IN =
      S_CWSP_S_NF,
      3,
      S_CWSP_S_CWIN_S_PGQ
         THRESHOLD = (S_CWSP_S_CWIN_S_NX * 3)
   PRIM_OUT = [1..3]GATHERED_OUTPUT_1
```

```
   PIP_OUT =
      S_CWSP_S_CWIN_S_EWCT
         VARIABLE PRODUCE = (S_CWSP_S_CWIN_S_EWCT + S_CWSP_S_CWIN_S_NX),
   S_SC1O
      VARIABLE PRODUCE = (S_CWSP_S_CWIN_S_EWCT + S_CWSP_S_CWIN_S_NX),
   S_CWSP_S_CWIN_S_EWCS
      PRODUCE = 1 OF 1
)

%ENDGRAPH
```

The Graph Value Set for the partition is empty:

```
%GV_SET
%END_SET
```

The autocoded source code for the mpid that implements the partition is:

```
/****************************************************************************/
/* File: p_cwsin_4.c                                                        */
/* Generated by the MCCI MPID Autocode Generator - Version: 0.9             */
/* On 10/26/98, at 20:36:52                                                 */
/* target: MERCURY_PPC  options: immed-write/N  probe/N  sid/N              */
/****************************************************************************/


/* Library */
#include "rts_sys.h"

/* Static Run-Time System Header File */
#include "srtshdrs.h"

/* Autocoded MPID Files */
#include "p_cwsin_4.constants.h"
#include "p_cwsin_4.in_neps.h"
#include "p_cwsin_4.mpid_data_type.h"
#include "p_cwsin_4.h"

void p_cwsin_4 (
   Persistent_Data_Type  *mpid_data,
   Rts_Handle_Type       rts_handle,
   int                   s_cwsp_s_cwin_s_ewq,
   int                   s_sclo,
   int                   s_cwsp_s_cwin_s_ewcs,
   int                   s_cwsp_s_cwin_s_ewrt,
   int                   gathered_output_1
      )
{
   char                  *s_cwsp_s_cwin_s_ewq_data_ptr [
   S_CWSP_S_CWIN_S_EWQ_MAX_FAMILY_SIZE];
   char                  *s_sclo_storage_ptrs [S_SC1O_MAX_FAMILY_SIZE];
   int                   s_sclo_produce_amount;
   char                  *s_cwsp_s_cwin_s_ewcs_storage_ptrs [
   S_CWSP_S_CWIN_S_EWCS_MAX_FAMILY_SIZE];
   int                   s_cwsp_s_cwin_s_ewcs_produce_amount;
   char                  *s_cwsp_s_cwin_s_ewrt_storage_ptrs [
   S_CWSP_S_CWIN_S_EWRT_MAX_FAMILY_SIZE];
   int                   s_cwsp_s_cwin_s_ewrt_produce_amount;
```

```
char                      *gathered_output_1_storage_ptrs [
GATHERED_OUTPUT_1_MAX_FAMILY_SIZE];
int                       gathered_output_1_produce_amount [
GATHERED_OUTPUT_1_MAX_FAMILY_SIZE];
int                       s_cwsp_s_nf;
float                     s_cwsp_s_cwin_s_f [1];
int                       s_cwsp_s_cwin_s_firsz1;
float                     s_cwsp_s_cwin_s_fir1 [11];
int                       s_cwsp_s_cwin_s_firsz2;
float                     s_cwsp_s_cwin_s_fir2 [39];
float                     s_cwsp_s_cwin_s_fs2;
int                       s_cwsp_s_cwin_s_nx;
int                       s_cwsp_s_cwin_s_isz;
int                       i1_p1_aasize;
float                     i1_p1_afi;
float                     i1_p1_afr;
int                       i1_p1_bsoffset;
int                       i1_p1_bsstep;
float                     i1_p1_fdelta;
float                     i1_p1_ffs;
float                     i1_p1_fm;
float                     i1_p1_fone;
float                     i1_p1_fround;
float                     i1_p1_fzero;
int                       i1_p1_mvalue;
float                     i1_p1_xf;
int                       i1_p1_xmode;
int                       i1_p5_xamt;
int                       i1_p5_ykmax;
int                       i1_p5_yvsize;
int index_mx;  /* Loop Parameter */
int index_n;  /* Loop Parameter */
int index_a;  /* Loop Parameter */
int index_ne;  /* Loop Parameter */

s_cwsp_s_nf = 20;
s_cwsp_s_cwin_s_f[0] = 2.44140600000000E-01;
s_cwsp_s_cwin_s_firsz1 = 11;
s_cwsp_s_cwin_s_fir1[0] = 9.87871740000000E-03;
s_cwsp_s_cwin_s_fir1[1] = - 4.20027990000000E-04;
s_cwsp_s_cwin_s_fir1[2] = - 5.85211180000000E-02;
s_cwsp_s_cwin_s_fir1[3] = 1.07341250000000E-03;
s_cwsp_s_cwin_s_fir1[4] = 2.98794630000000E-01;
s_cwsp_s_cwin_s_fir1[5] = 4.98531370000000E-01;
s_cwsp_s_cwin_s_fir1[6] = 2.98794630000000E-01;
s_cwsp_s_cwin_s_fir1[7] = 1.07341250000000E-03;
s_cwsp_s_cwin_s_fir1[8] = - 5.85211180000000E-02;
s_cwsp_s_cwin_s_fir1[9] = - 4.20027990000000E-04;
s_cwsp_s_cwin_s_fir1[10] = 9.87871740000000E-03;
s_cwsp_s_cwin_s_firsz2 = 39;
s_cwsp_s_cwin_s_fir2[0] = - 1.23575920000000E-03;
s_cwsp_s_cwin_s_fir2[1] = 8.70276560000000E-05;
s_cwsp_s_cwin_s_fir2[2] = 2.53144400000000E-03;
s_cwsp_s_cwin_s_fir2[3] = - 7.23021220000000E-05;
s_cwsp_s_cwin_s_fir2[4] = - 4.79434850000000E-03;
s_cwsp_s_cwin_s_fir2[5] = 1.93636250000000E-04;
s_cwsp_s_cwin_s_fir2[6] = 8.44822920000000E-03;
s_cwsp_s_cwin_s_fir2[7] = - 2.12764510000000E-04;
s_cwsp_s_cwin_s_fir2[8] = - 1.37729710000000E-02;
s_cwsp_s_cwin_s_fir2[9] = 3.47536320000000E-04;
```

```
s_cwsp_s_cwin_s_fir2[10] = 2.18234450000000E-02;
s_cwsp_s_cwin_s_fir2[11] = - 3.78957080000000E-04;
s_cwsp_s_cwin_s_fir2[12] = - 3.40838430000000E-02;
s_cwsp_s_cwin_s_fir2[13] = 5.03258610000000E-04;
s_cwsp_s_cwin_s_fir2[14] = 5.51007170000000E-02;
s_cwsp_s_cwin_s_fir2[15] = - 5.15613120000000E-04;
s_cwsp_s_cwin_s_fir2[16] = - 1.00731690000000E-01;
s_cwsp_s_cwin_s_fir2[17] = 6.01775770000000E-04;
s_cwsp_s_cwin_s_fir2[18] = 3.16516400000000E-01;
s_cwsp_s_cwin_s_fir2[19] = 4.99431310000000E-01;
s_cwsp_s_cwin_s_fir2[20] = 3.16516400000000E-01;
s_cwsp_s_cwin_s_fir2[21] = 6.01775770000000E-04;
s_cwsp_s_cwin_s_fir2[22] = - 1.00731690000000E-01;
s_cwsp_s_cwin_s_fir2[23] = - 5.15613120000000E-04;
s_cwsp_s_cwin_s_fir2[24] = 5.51007170000000E-02;
s_cwsp_s_cwin_s_fir2[25] = 5.03258610000000E-04;
s_cwsp_s_cwin_s_fir2[26] = - 3.40838430000000E-02;
s_cwsp_s_cwin_s_fir2[27] = - 3.78957080000000E-04;
s_cwsp_s_cwin_s_fir2[28] = 2.18234450000000E-02;
s_cwsp_s_cwin_s_fir2[29] = 3.47536320000000E-04;
s_cwsp_s_cwin_s_fir2[30] = - 1.37729710000000E-02;
s_cwsp_s_cwin_s_fir2[31] = - 2.12764510000000E-04;
s_cwsp_s_cwin_s_fir2[32] = 8.44822920000000E-03;
s_cwsp_s_cwin_s_fir2[33] = 1.93636250000000E-04;
s_cwsp_s_cwin_s_fir2[34] = - 4.79434850000000E-03;
s_cwsp_s_cwin_s_fir2[35] = - 7.23021220000000E-05;
s_cwsp_s_cwin_s_fir2[36] = 2.53144400000000E-03;
s_cwsp_s_cwin_s_fir2[37] = 8.70276560000000E-05;
s_cwsp_s_cwin_s_fir2[38] = - 1.23575920000000E-03;
s_cwsp_s_cwin_s_fs2 = 1.00000000000000E+00;
s_cwsp_s_cwin_s_nx = 400;
s_cwsp_s_cwin_s_isz = 1600;
read_queue_srts (
   s_cwsp_s_cwin_s_ewq,
   s_cwsp_s_cwin_s_ewq_read_amount[mpid_data->state][0],
   s_cwsp_s_cwin_s_ewq_offset_amount[mpid_data->state][0],
   &s_cwsp_s_cwin_s_ewq_data_ptr[0],
   rts_handle
   );
i1_p1_fone = 1.00000000000000E+00;
i1_p1_fround = 5.00000000000000E-01;
i1_p1_fzero = 0.00000000000000E+00;
i1_p1_mvalue = 1024;
vfilli (
   &mpid_data->persistent_area[58712],
   &mpid_data->scratch_area[8256],
   1,
   3
   );
i1_p1_xmode = 1;
memcpy_mcci (
   &mpid_data->scratch_area[8224],
   1,
   &s_cwsp_s_cwin_s_f,
   1,
   4,
   1
   );
vfill (
   &mpid_data->scratch_area[8224],
```

```
      &mpid_data->scratch_area[8224] + 4,
      1,
      2
      );
   i1_p1_ffs = 1.00000000000000E+00;
   to_float_mcci (
      &i1_p1_mvalue,
      dint,
      0,
      &i1_p1_fm,
      0
      );
   i1_p1_fdelta = -6.28318530720000E+00 / i1_p1_fm;
   i1_p1_aasize = 8;
   vramp (
      &i1_p1_fzero,
      &i1_p1_fdelta,
      &mpid_data->scratch_area[0],
      2,
      i1_p1_mvalue
      );
   cvexp (
      &mpid_data->scratch_area[0],
      2,
      &mpid_data->scratch_area[0],
      2,
      i1_p1_mvalue
      );
   vabs (
      &mpid_data->scratch_area[8224],
      1,
      &mpid_data->scratch_area[8208],
      1,
      3
      );
   lveq (
      &mpid_data->scratch_area[8224],
      1,
      &mpid_data->scratch_area[8208],
      1,
      &mpid_data->scratch_area[8272],
      1,
      3
      );
   vlim (
      &mpid_data->scratch_area[8272],
      1,
      &i1_p1_fone,
      &i1_p1_fone,
      &mpid_data->scratch_area[8272],
      1,
      3
      );
   vsmul (
      &mpid_data->scratch_area[8208],
      1,
      &i1_p1_fm,
      &mpid_data->scratch_area[8208],
      1,
      3
```

```
    );
vsdiv (
    &mpid_data->scratch_area[8208],
    1,
    &i1_p1_ffs,
    &mpid_data->scratch_area[8208],
    1,
    3
    );
vsadd (
    &mpid_data->scratch_area[8208],
    1,
    &i1_p1_fround,
    &mpid_data->scratch_area[8208],
    1,
    3
    );
vmul (
    &mpid_data->scratch_area[8208],
    1,
    &mpid_data->scratch_area[8272],
    1,
    &mpid_data->scratch_area[8208],
    1,
    3
    );
vfix32 (
    &mpid_data->scratch_area[8208],
    1,
    &mpid_data->scratch_area[8240],
    1,
    3
    );
memcpy_mcci (
    &i1_p1_fdelta,
    1,
    &mpid_data->scratch_area[8224],
    1,
    4,
    1
    );
if ((is_equal_mcci (i1_p1_fdelta, 0.00000000000000E+00)))
{
    /* Error recovery NOT implemented (as of 8.7.95). */
    /* BEGIN removed message...
    S_CWSP_S_CWIN_S_F must not be 0
    ...END removed message */
}
else
    if ((is_greaterthan_mcci (i1_p1_fdelta, i1_p1_ffs)))
    {
        /* Error recovery NOT implemented (as of 8.7.95). */
        /* BEGIN removed message...
        S_CWSP_S_CWIN_S_F greater than sampling frequency
        ...END removed message */
    }
memcpy_mcci (
    &i1_p1_xmode,
    1,
    &mpid_data->scratch_area[8256],
```

```
        1,
        4,
        1
        );
    if (((i1_p1_xmode < 0) || (i1_p1_xmode >= i1_p1_mvalue)))
    {
        /* Error recovery NOT implemented (as of 8.7.95). */
        /* BEGIN removed message...
        Bandshift table pointer is out of range
        ...END removed message */
    }
    vsmuli (
        &mpid_data->scratch_area[8256],
        1,
        &i1_p1_aasize,
        &mpid_data->scratch_area[8192],
        1,
        3
        );
    for (index_mx = 0; index_mx <= 2; index_mx++)
    {
        memcpy_mcci (
            &i1_p1_bsoffset,
            1,
            &mpid_data->scratch_area[8192] + index_mx * 4,
            1,
            4,
            1
            );
        memcpy_mcci (
            &i1_p1_bsstep,
            1,
            &mpid_data->scratch_area[8240] + index_mx * 4,
            1,
            4,
            1
            );
        for (index_n = 0; index_n <= 1599; index_n++)
        {
            memcpy_mcci (
                &i1_p1_xf,
                1,
                s_cwsp_s_cwin_s_ewq_data_ptr[0] + 4 * (index_n * 3 + index_mx) +
                0 * index_n * 4,
                1,
                4,
                1
                );
            memcpy_mcci (
                &i1_p1_afr,
                1,
                &mpid_data->scratch_area[0] + i1_p1_bsoffset,
                1,
                4,
                1
                );
            memcpy_mcci (
                &i1_p1_afi,
                1,
                &mpid_data->scratch_area[0] + i1_p1_bsoffset + 4,
```

```
         1,
         4,
         1
         );
      i1_p1_afr = i1_p1_afr * i1_p1_xf;
      i1_p1_afi = i1_p1_afi * i1_p1_xf;
      memcpy_mcci (
         &mpid_data->persistent_area[20304] + (index_n * 3 + index_mx) *
         i1_p1_aasize,
         1,
         &i1_p1_afr,
         1,
         4,
         1
         );
      memcpy_mcci (
         &mpid_data->persistent_area[20304] + (index_n * 3 + index_mx) *
         i1_p1_aasize + 4,
         1,
         &i1_p1_afi,
         1,
         4,
         1
         );
      i1_p1_bsoffset = i1_p1_bsoffset + i1_p1_aasize * i1_p1_bsstep;
      if ((i1_p1_bsoffset >= i1_p1_mvalue * i1_p1_aasize))
      {
         i1_p1_bsoffset = i1_p1_bsoffset - i1_p1_mvalue * i1_p1_aasize;
      }
      else
      {
         if (i1_p1_bsoffset < 0)
         {
            i1_p1_bsoffset = i1_p1_bsoffset + i1_p1_mvalue * i1_p1_aasize;
         }
      }
   }
   memcpy_mcci (
      &i1_p1_bsoffset,
      1,
      &mpid_data->scratch_area[8256] + index_mx * 4,
      1,
      4,
      1
      );
   if ((i1_p1_bsstep > 0))
   {
      i1_p1_bsoffset = i1_p1_bsoffset + 1600 * i1_p1_bsstep;
      while ((i1_p1_bsoffset >= i1_p1_mvalue))
      {
         i1_p1_bsoffset = i1_p1_bsoffset - i1_p1_mvalue;
      }
   }
   else
   {
      i1_p1_bsoffset = i1_p1_bsoffset + 1600 * i1_p1_bsstep;
      while ((i1_p1_bsoffset < 0))
      {
         i1_p1_bsoffset = i1_p1_bsoffset + i1_p1_mvalue;
      }
```

```
    }
    memcpy_mcci (
        &mpid_data->scratch_area[8256] + index_mx * 4,
        1,
        &i1_p1_bsoffset,
        1,
        4,
        1
        );
}
vmovi (
    &mpid_data->scratch_area[8256],
    1,
    &mpid_data->persistent_area[58716],
    1,
    1
    );
vmov (
    &s_cwsp_s_cwin_s_fir1,
    1,
    &mpid_data->scratch_area[0],
    1,
    11
    );
for (index_mx = 0; index_mx <= 2; index_mx++)
{
    vsmul (
        &mpid_data->persistent_area[20088] + index_mx * 8,
        12,
        &mpid_data->scratch_area[0] + 40,
        &mpid_data->persistent_area[888] + index_mx * 8,
        6,
        800
        );
    vsmul (
        &mpid_data->persistent_area[20088] + index_mx * 8 + 4,
        12,
        &mpid_data->scratch_area[0] + 40,
        &mpid_data->persistent_area[888] + index_mx * 8 + 4,
        6,
        800
        );
    for (index_a = 1; index_a <= 10; index_a++)
    {
        vma (
            &mpid_data->persistent_area[20088] + (index_a * 3 + index_mx) * 8,

            12,
            &mpid_data->scratch_area[0] + (- index_a + 10) * 4,
            0,
            &mpid_data->persistent_area[888] + index_mx * 8,
            6,
            &mpid_data->persistent_area[888] + index_mx * 8,
            6,
            800
            );
    }
    for (index_a = 1; index_a <= 10; index_a++)
    {
        vma (
```

```
            &mpid_data->persistent_area[20088] + (index_a * 3 + index_mx) * 8
            + 4,
            12,
            &mpid_data->scratch_area[0] + (- index_a + 10) * 4,
            0,
            &mpid_data->persistent_area[888] + index_mx * 8 + 4,
            6,
            &mpid_data->persistent_area[888] + index_mx * 8 + 4,
            6,
            800
            );
    }
}
vmov (
    &s_cwsp_s_cwin_s_fir2,
    1,
    &mpid_data->scratch_area[9600],
    1,
    39
    );
for (index_mx = 0; index_mx <= 2; index_mx++)
{
    vsmul (
        &mpid_data->persistent_area[0] + index_mx * 8,
        12,
        &mpid_data->scratch_area[9600] + 152,
        &mpid_data->scratch_area[0] + index_mx * 8,
        6,
        400
        );
    vsmul (
        &mpid_data->persistent_area[0] + index_mx * 8 + 4,
        12,
        &mpid_data->scratch_area[9600] + 152,
        &mpid_data->scratch_area[0] + index_mx * 8 + 4,
        6,
        400
        );
    for (index_a = 1; index_a <= 38; index_a++)
    {
        vma (
            &mpid_data->persistent_area[0] + (index_a * 3 + index_mx) * 8,
            12,
            &mpid_data->scratch_area[9600] + (- index_a + 38) * 4,
            0,
            &mpid_data->scratch_area[0] + index_mx * 8,
            6,
            &mpid_data->scratch_area[0] + index_mx * 8,
            6,
            400
            );
    }
    for (index_a = 1; index_a <= 38; index_a++)
    {
        vma (
            &mpid_data->persistent_area[0] + (index_a * 3 + index_mx) * 8 + 4,

            12,
            &mpid_data->scratch_area[9600] + (- index_a + 38) * 4,
            0,
```

```
            &mpid_data->scratch_area[0] + index_mx * 8 + 4,
            6,
            &mpid_data->scratch_area[0] + index_mx * 8 + 4,
            6,
            400
            );
      }
   }
   cvmov (
      &mpid_data->scratch_area[0],
      2,
      &mpid_data->scratch_area[48076],
      2,
      1200
      );
   i1_p5_ykmax = 20;
   i1_p5_yvsize = 160;
   for (index_ne = 0; index_ne <= 19; index_ne++)
   {
      i1_p5_xamt = 20;
      if ((20 > i1_p5_ykmax))
      {
         i1_p5_xamt = i1_p5_ykmax;
      }
      cvmov (
         &mpid_data->scratch_area[48076] + index_ne * 480,
         6,
         &mpid_data->scratch_area[16] + index_ne * i1_p5_yvsize,
         2,
         i1_p5_xamt
         );
      if ((i1_p5_xamt < 20))
      {
         vclr (
            &mpid_data->scratch_area[16] + index_ne * i1_p5_yvsize +
            i1_p5_xamt * 8,
            1,
            (- i1_p5_xamt + 20) * 2
            );
      }
      if ((index_ne == 0))
      {
      }
      if ((i1_p5_xamt > i1_p5_ykmax))
      {
         i1_p5_xamt = i1_p5_ykmax;
      }
      cvmov (
         &mpid_data->scratch_area[48076] + index_ne * 480 + 8,
         6,
         &mpid_data->scratch_area[3216] + index_ne * i1_p5_yvsize,
         2,
         i1_p5_xamt
         );
      if ((i1_p5_xamt < 20))
      {
         vclr (
            &mpid_data->scratch_area[3216] + index_ne * i1_p5_yvsize +
            i1_p5_xamt * 8,
            1,
```

```
            (- i1_p5_xamt + 20) * 2
            );
      }
      if ((index_ne == 0))
      {
      }
      if ((i1_p5_xamt > i1_p5_ykmax))
      {
         i1_p5_xamt = i1_p5_ykmax;
      }
      cvmov (
         &mpid_data->scratch_area[48076] + index_ne * 480 + 16,
         6,
         &mpid_data->scratch_area[6416] + index_ne * i1_p5_yvsize,
         2,
         i1_p5_xamt
         );
      if ((i1_p5_xamt < 20))
      {
         vclr (
            &mpid_data->scratch_area[6416] + index_ne * i1_p5_yvsize +
            i1_p5_xamt * 8,
            1,
            (- i1_p5_xamt + 20) * 2
            );
      }
      if ((index_ne == 0))
      {
      }
   }
   *((int *)&mpid_data->persistent_area[58708]) = *((int *)&mpid_data->
   persistent_area[58704]) + s_cwsp_s_cwin_s_nx;
   *((int *)&mpid_data->scratch_area[0]) = *((int *)&mpid_data->
   persistent_area[58704]) + s_cwsp_s_cwin_s_nx;
   *((int *)&mpid_data->scratch_area[8]) = 1;
   copy_data_srts (
      &mpid_data->persistent_area[19200],
      &mpid_data->persistent_area[0],
      888
      );
   copy_data_srts (
      &mpid_data->persistent_area[58488],
      &mpid_data->persistent_area[20088],
      216
      );
   copy_data_srts (
      &mpid_data->persistent_area[58708],
      &mpid_data->persistent_area[58704],
      4
      );
   copy_data_srts (
      &mpid_data->persistent_area[58716],
      &mpid_data->persistent_area[58712],
      4
      );
   s_sclo_produce_amount = 1;
   s_sclo_storage_ptrs[0] = &mpid_data->scratch_area[0];
   write_queue_srts (
      s_sclo,
      s_sclo_produce_amount,
```

```
      s_sclo_storage_ptrs[0],
      0,
      rts_handle
      );
   s_cwsp_s_cwin_s_ewcs_produce_amount = 1;
   s_cwsp_s_cwin_s_ewcs_storage_ptrs[0] = &mpid_data->scratch_area[8];
   write_queue_srts (
      s_cwsp_s_cwin_s_ewcs,
      s_cwsp_s_cwin_s_ewcs_produce_amount,
      s_cwsp_s_cwin_s_ewcs_storage_ptrs[0],
      0,
      rts_handle
      );
   s_cwsp_s_cwin_s_ewrt_produce_amount = 1200;
   s_cwsp_s_cwin_s_ewrt_storage_ptrs[0] = &mpid_data->scratch_area[48076];
   write_queue_srts (
      s_cwsp_s_cwin_s_ewrt,
      s_cwsp_s_cwin_s_ewrt_produce_amount,
      s_cwsp_s_cwin_s_ewrt_storage_ptrs[0],
      0,
      rts_handle
      );
   gathered_output_1_produce_amount[0] = 400;
   gathered_output_1_produce_amount[1] = 400;
   gathered_output_1_produce_amount[2] = 400;
   gathered_output_1_storage_ptrs[0] = &mpid_data->scratch_area[16];
   gathered_output_1_storage_ptrs[1] = &mpid_data->scratch_area[3216];
   gathered_output_1_storage_ptrs[2] = &mpid_data->scratch_area[6416];
   write_queue_family_srts (
      gathered_output_1,
      gathered_output_1_produce_amount,
      gathered_output_1_storage_ptrs,
      0,
      rts_handle
      );
   consume_queue_srts (
      s_cwsp_s_cwin_s_ewq,
      s_cwsp_s_cwin_s_ewq_consume_amount[mpid_data->state][0],
      rts_handle
      );
}

void p_cwsin_4_reinit_local_info (
   Persistent_Data_Type   *mpid_data,
   Rts_Handle_Type        rts_handle
      )
{
   cfloat                 s_cwsp_s_cwin_s_ewl2_init_array;
   cfloat                 s_cwsp_s_cwin_s_ewbs_init_array;
   int                    s_cwsp_s_cwin_s_ewct_init_array;
   int                    s_cwsp_s_cwin_s_npew_init_array;

   mpid_data->state = TOP_OF_PERIOD;
   if (mpid_data->persistent_area == SRTS_NULL)
   {
      mpid_data->persistent_area = (char *)SRTS_MEM_aligned_malloc (
      SRTS_MEM_MPIDPRIVATE, 58720, 0, rts_handle);
   }
   if (mpid_data->scratch_area == SRTS_NULL)
   {
```

```
      mpid_data->scratch_area = (char *)SRTS_MEM_aligned_malloc (
      SRTS_MEM_MPIDSCRATCH, 57676, 0, rts_handle);
   }
   s_cwsp_s_cwin_s_ew12_init_array.real = 0.00000000000000E+00;
   s_cwsp_s_cwin_s_ew12_init_array.imag = 0.00000000000000E+00;
   init_buff_srts (
      &mpid_data->persistent_area[0],
      (char *)&s_cwsp_s_cwin_s_ew12_init_array,
      sizeof (cfloat),
      111
      );
   s_cwsp_s_cwin_s_ewbs_init_array.real = 0.00000000000000E+00;
   s_cwsp_s_cwin_s_ewbs_init_array.imag = 0.00000000000000E+00;
   init_buff_srts (
      &mpid_data->persistent_area[20088],
      (char *)&s_cwsp_s_cwin_s_ewbs_init_array,
      sizeof (cfloat),
      27
      );
   s_cwsp_s_cwin_s_ewct_init_array = 0;
   init_buff_srts (
      &mpid_data->persistent_area[58704],
      (char *)&s_cwsp_s_cwin_s_ewct_init_array,
      sizeof (int),
      1
      );
   s_cwsp_s_cwin_s_npew_init_array = 0;
   init_buff_srts (
      &mpid_data->persistent_area[58712],
      (char *)&s_cwsp_s_cwin_s_npew_init_array,
      sizeof (int),
      1
      );
}

void p_cwsin_4_det_gv_set (
   Persistent_Data_Type  *mpid_data,
   Rts_Handle_Type       rts_handle
      )
{
   mpid_data->state = 1;
}

void *p_cwsin_4_cleanup (
   void                  *arg_ptr,
   Rts_Handle_Type       rts_handle
      )
{
   Persistent_Data_Type  *mpid_data = (Persistent_Data_Type *)arg_ptr;

   if (mpid_data->persistent_area)
   {
      SRTS_MEM_free (
         SRTS_MEM_MPIDPRIVATE,
         mpid_data->persistent_area,
         58720,
         rts_handle
         );
      mpid_data->persistent_area = SRTS_NULL;
   }
```

```
    if (mpid_data->scratch_area)
    {
        SRTS_MEM_free (
            SRTS_MEM_MPIDSCRATCH,
            mpid_data->scratch_area,
            57676,
            rts_handle
            );
        mpid_data->scratch_area = SRTS_NULL;
    }
    return (
        SRTS_NULL
        );
}
```

The iconic form for each of the partitions is contained in Appendix D.

6.4.5 Testing

Complete testing of the converted graph was beyond the scope of this effort. Additionally, test vectors were not available. However, some of the partitions were individually tested. The following description is representative of the testing performed.

Simulated sensor data set generated for the CWS mode is shown in Figures 11 through Figure 13. This data was then used in the MPID Test Environment to individually exercise each autocoded partition. The output from the partition was used as the input to the downstream partition. Values of GIPs and VARs required to execute the mpid were entered into data files. A script file was constructed to execute the graph by executing the partitions in a "control flow" order that was derived from the data flow graph.

A modified version of the Equivalent Application Graph is shown in Figure 14. The graph contains only those equivalent nodes (i.e., partitions) that are executed when the CWS mode is being processed. The bolded lines in the figure indicate queues for which plots of queue contents were captured and these plots are included as figures in this document.

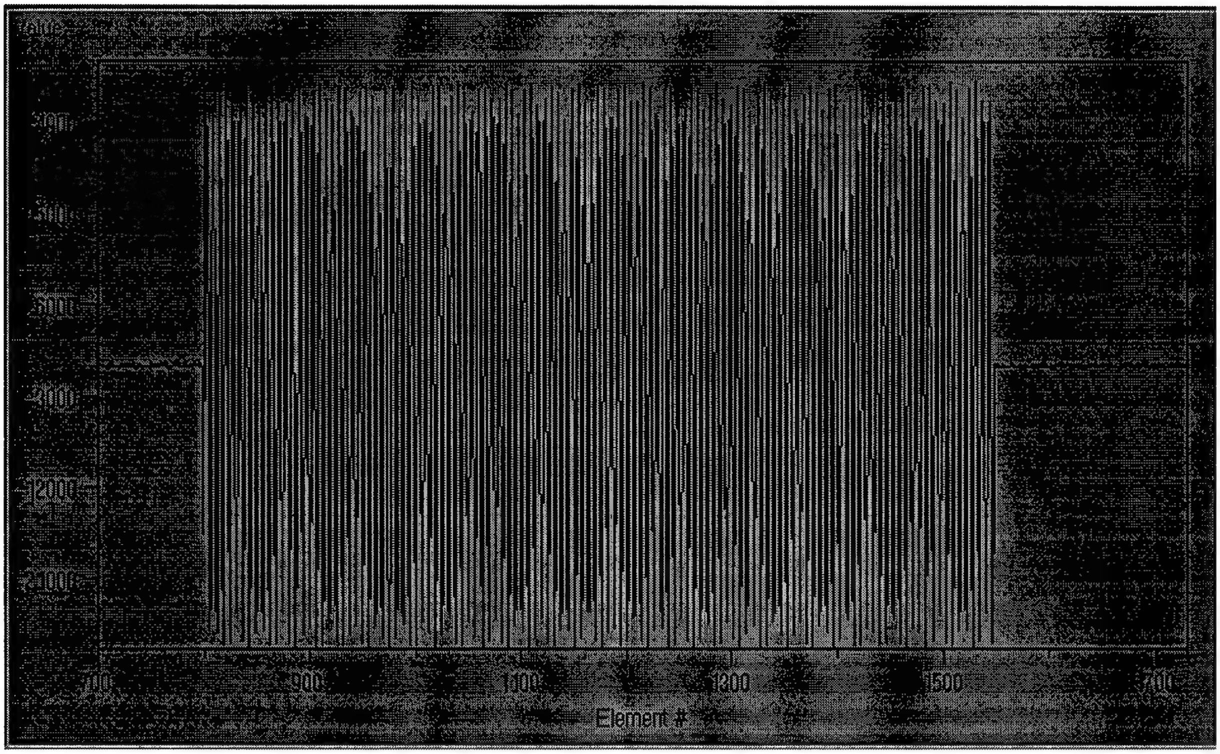


Figure 11. Simulated EW Sensor Data

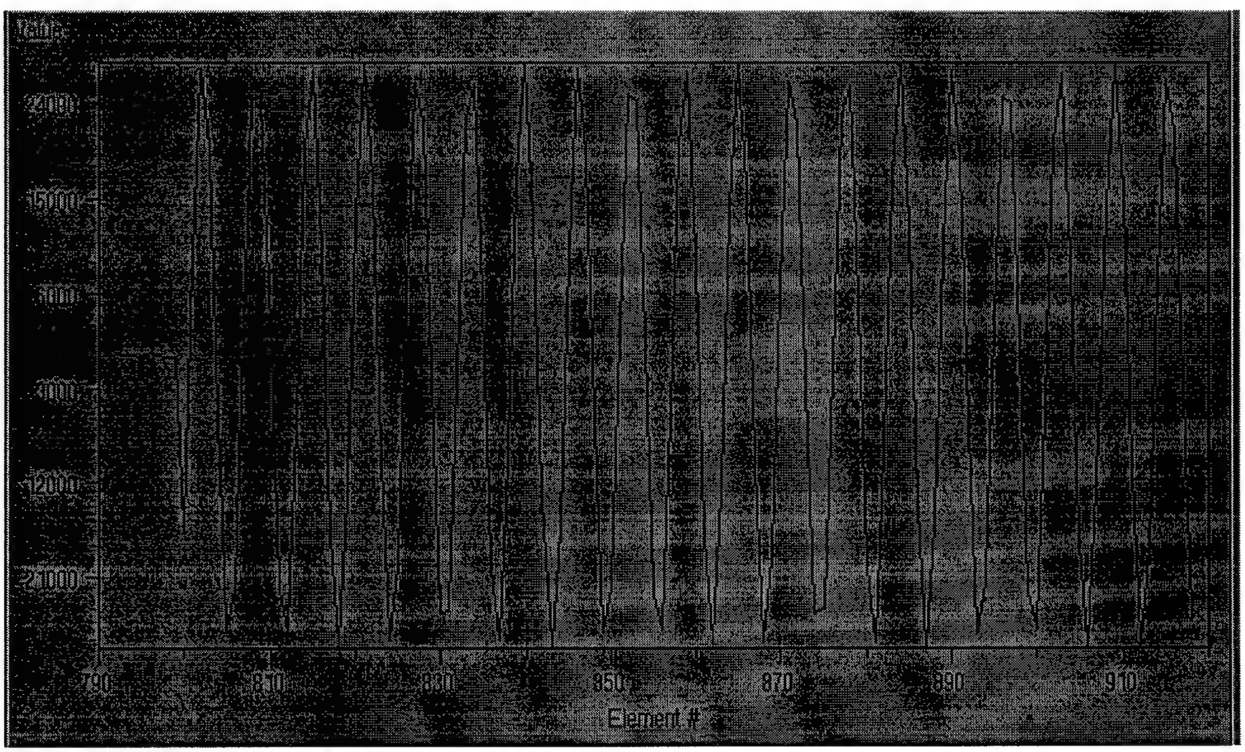


Figure 12. Detail of Simulated EW Sensor Data

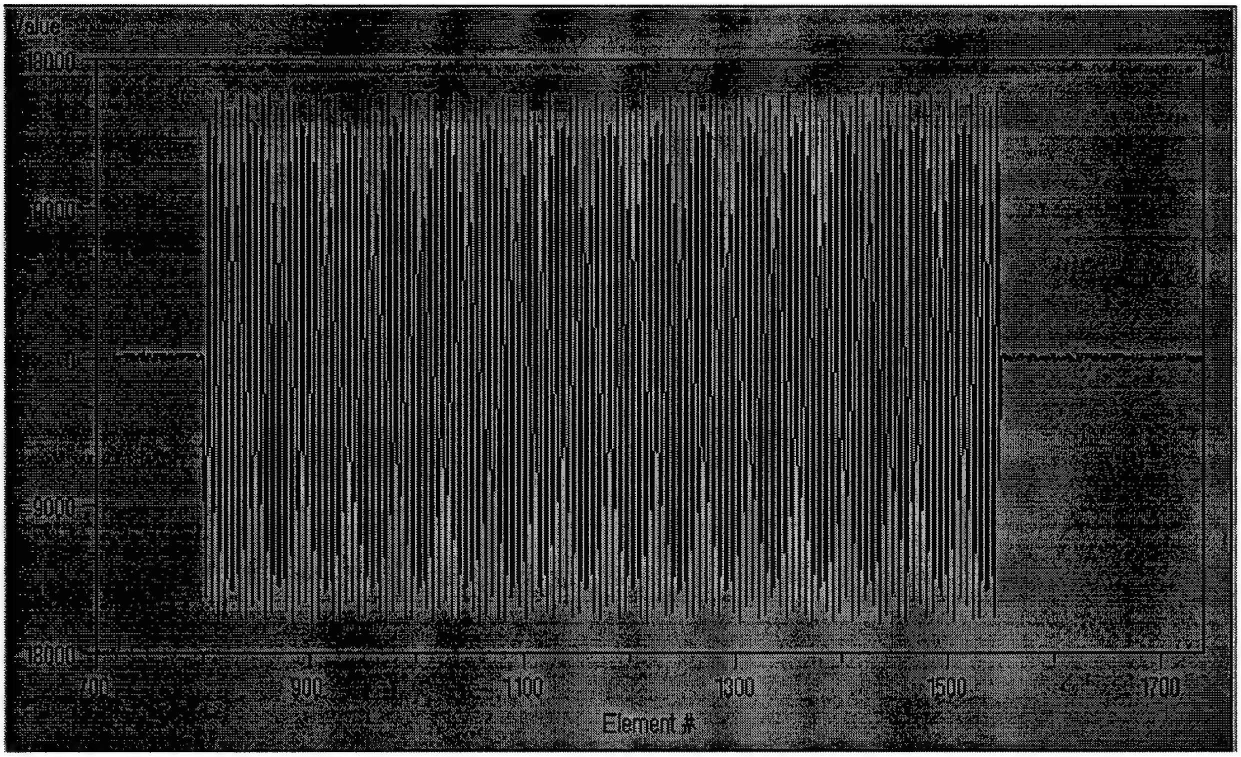


Figure 13. Simulated NS Sensor Data

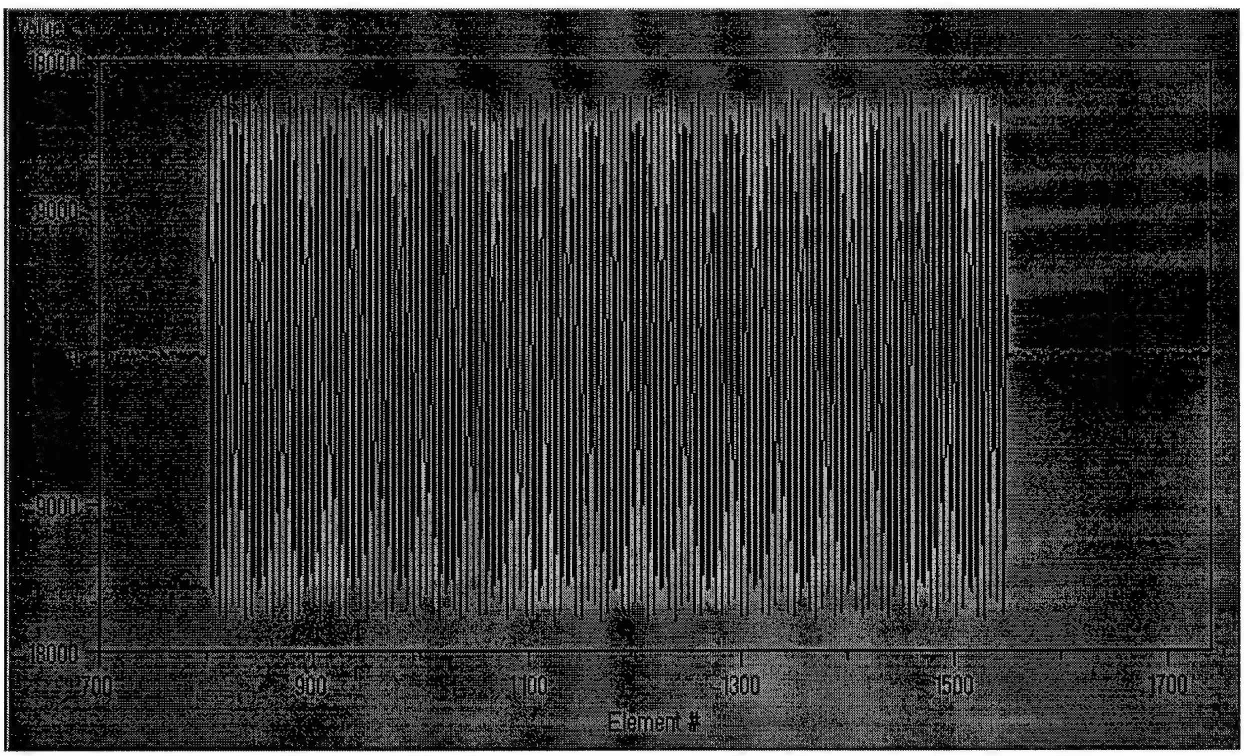


Figure 14. Simulated Omni Sensor Data

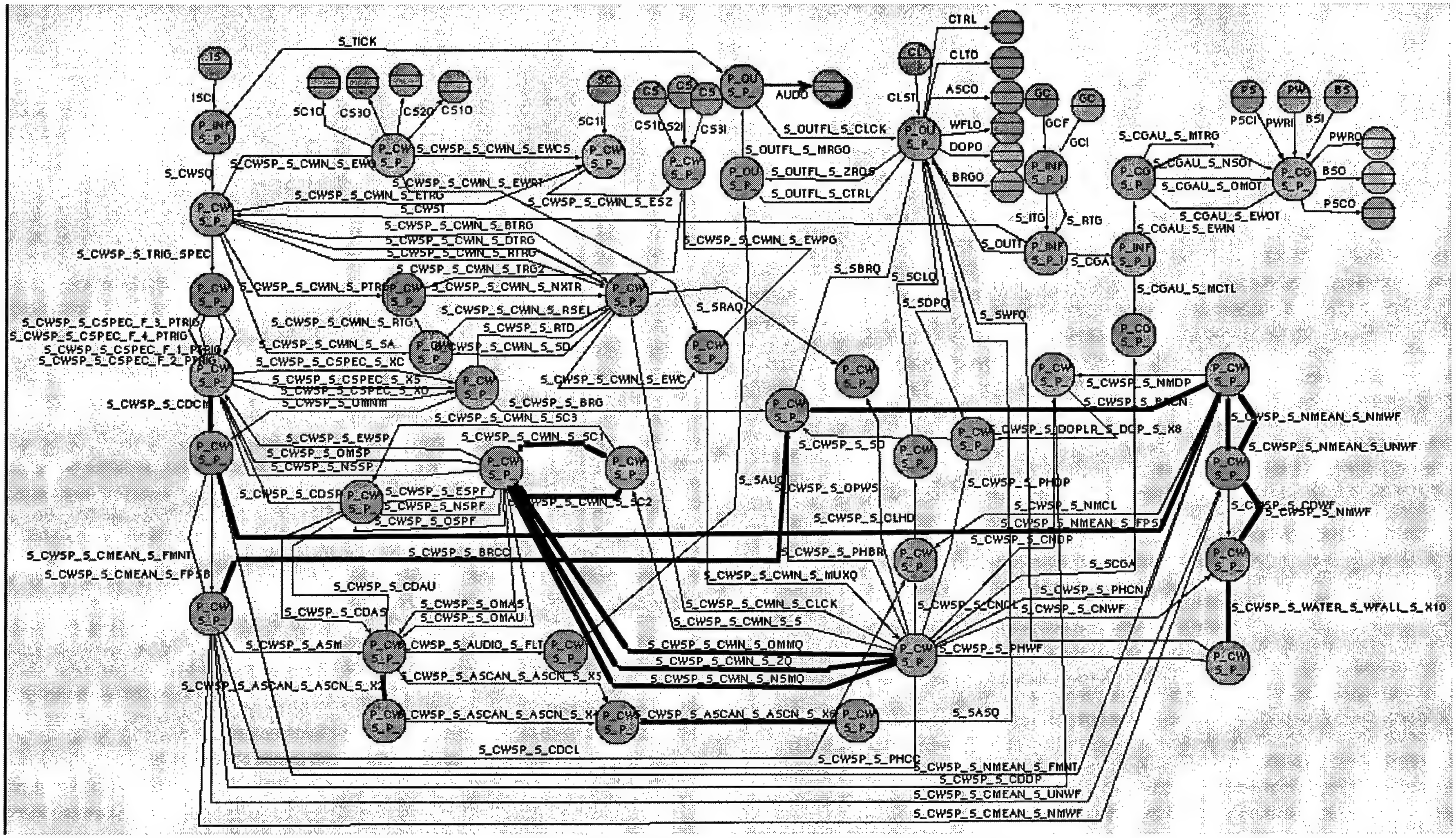


Figure 15. Modified DICASS Equivalent Application Graph

(contains only those nodes that execute as part of mode CWS)

Representative data from one test set is shown in the following figures. Since actual parameter values were not available, many parameters were set to either one or zero. For other parameters either an educated or an uneducated guess was used.

The sensor data after basebanding and filtering is shown in Figures 16 through 18.

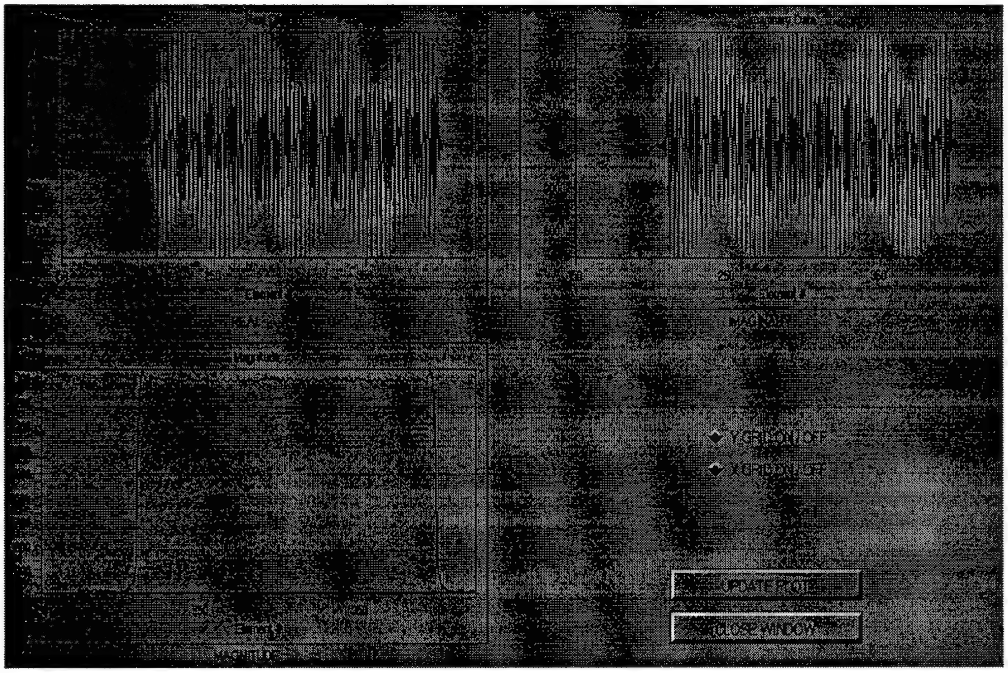


Figure 16. Data on Queue ZQ

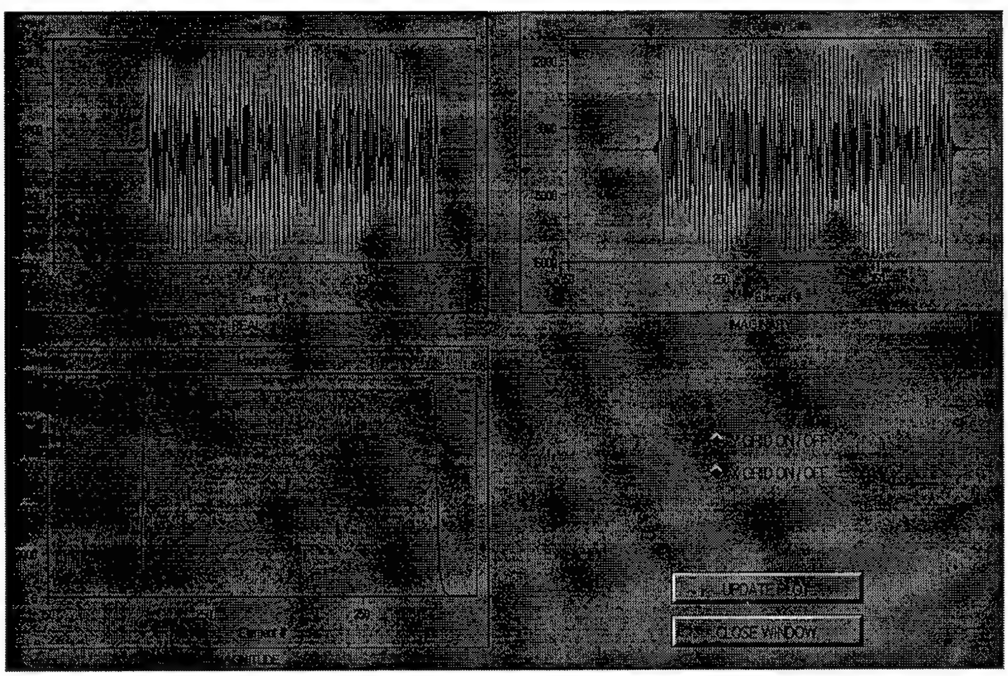


Figure 17. Data on Queue NSMQ

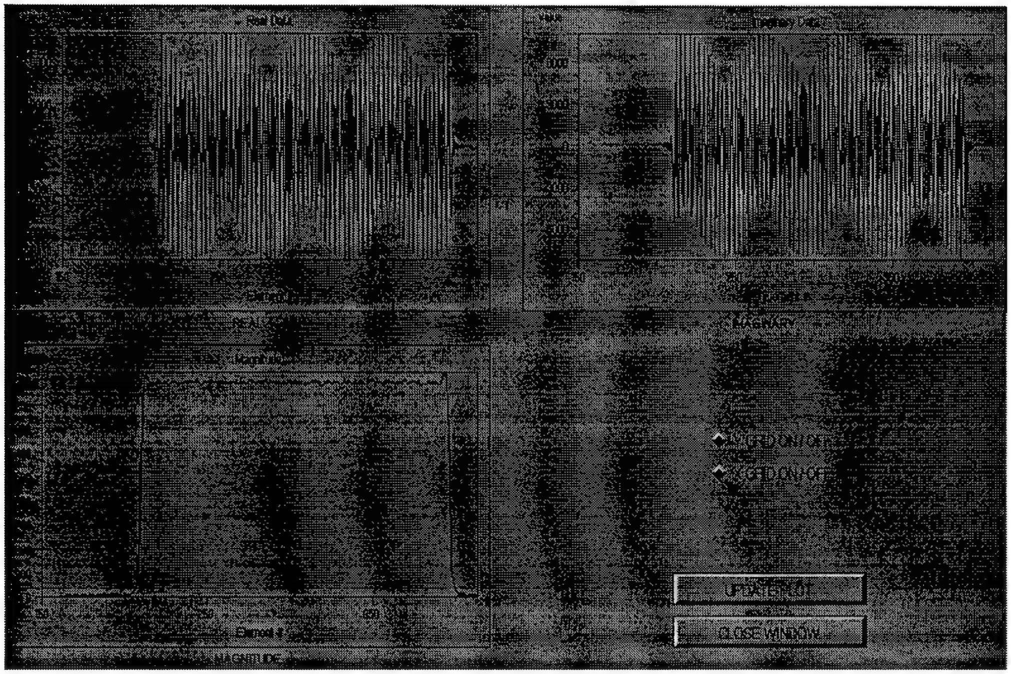


Figure 18. Data on Queue OMMQ

The following figures (19-32) show the contents of selected queues that connect equivalent nodes.

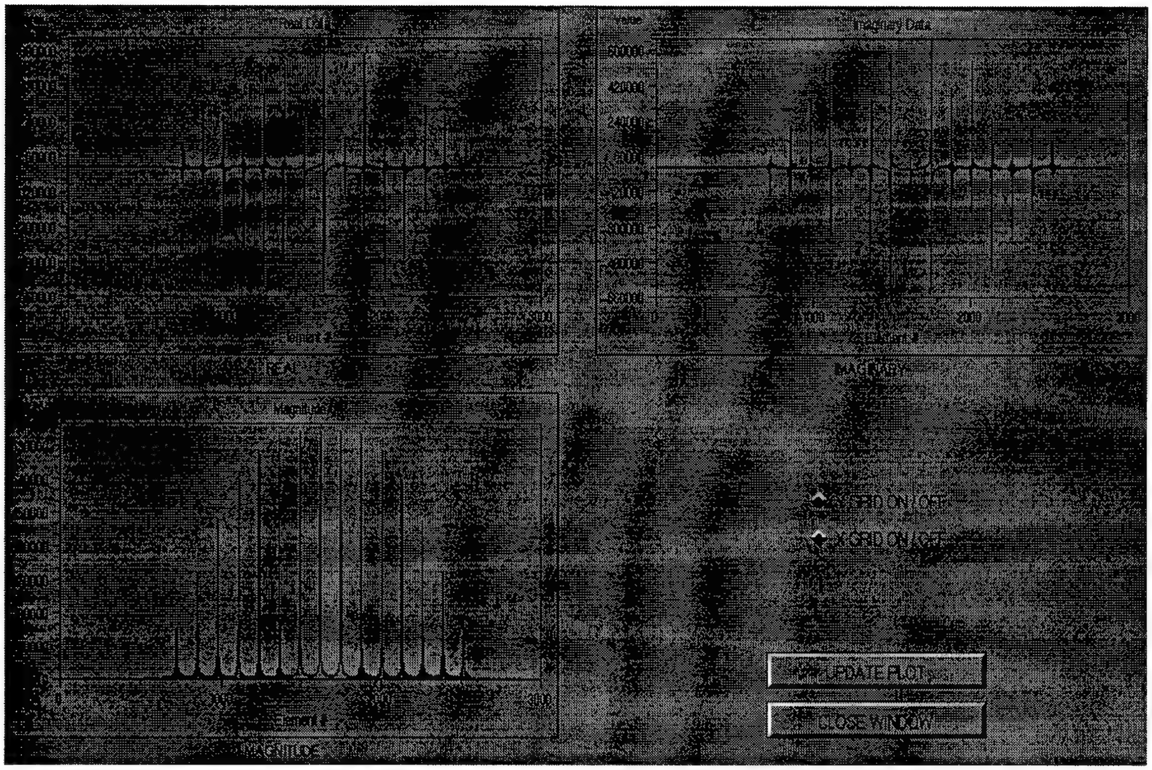


Figure 19. Data on Queue SC1

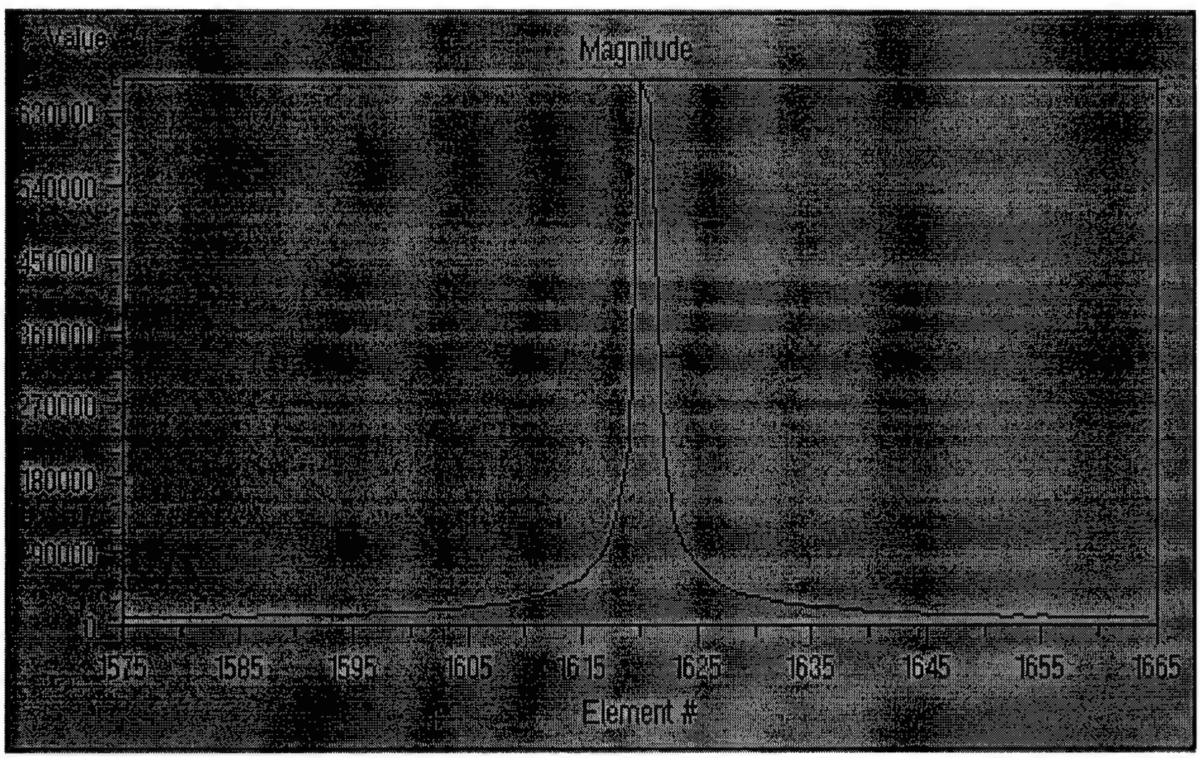


Figure 20. Detail of Data on Queue SC1

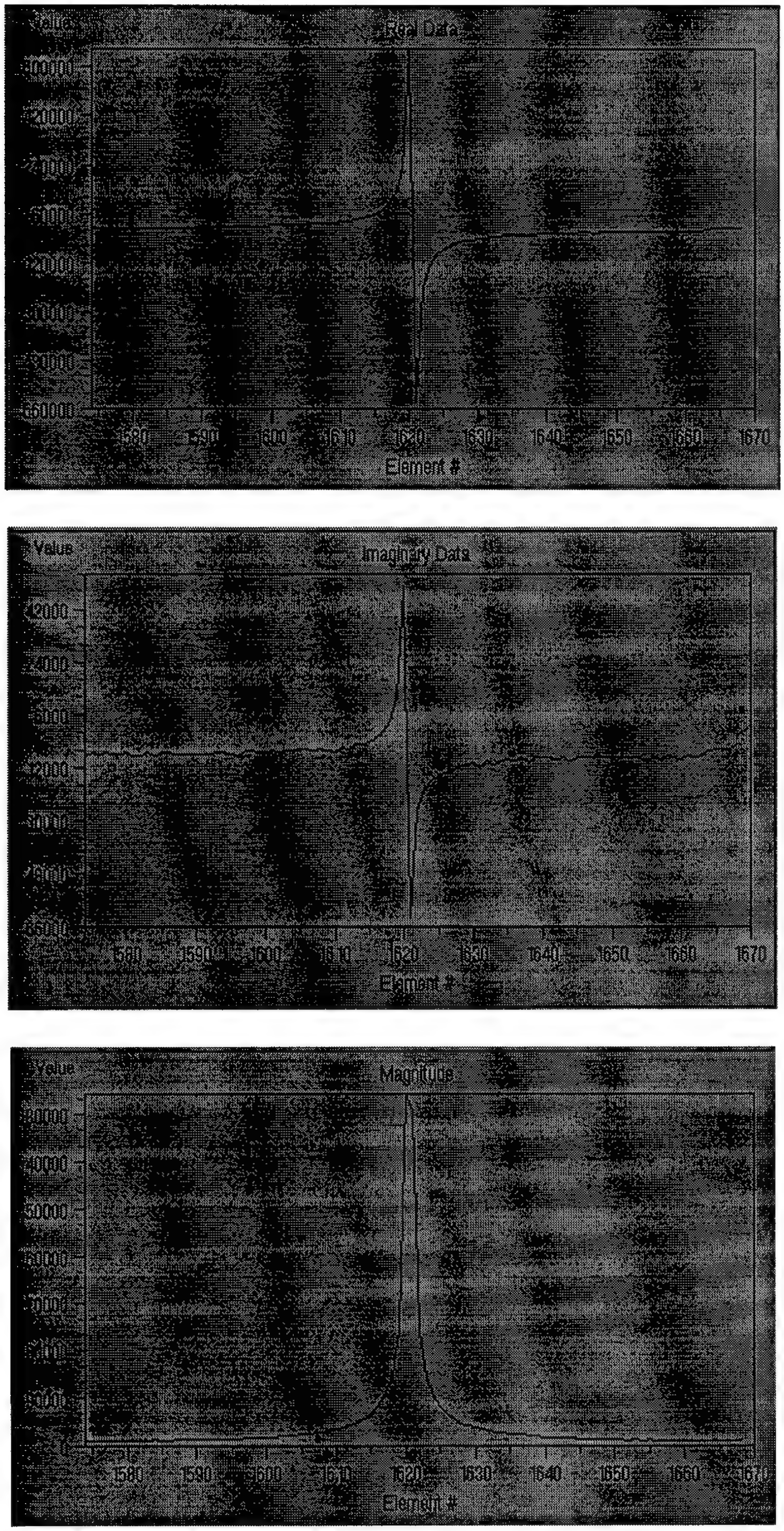


Figure 21. Detail of Data on SC2 - Real, Imaginary, and Magnitude

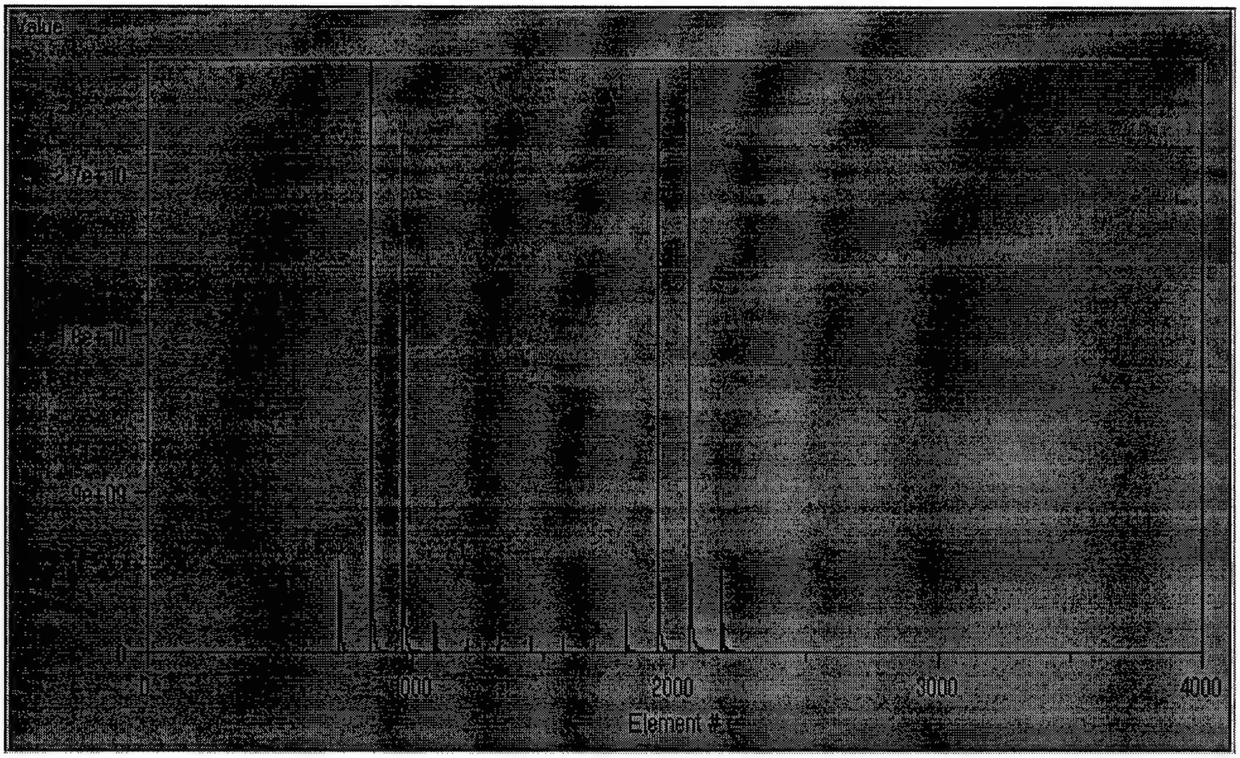


Figure 22. Data on Queue CDCM

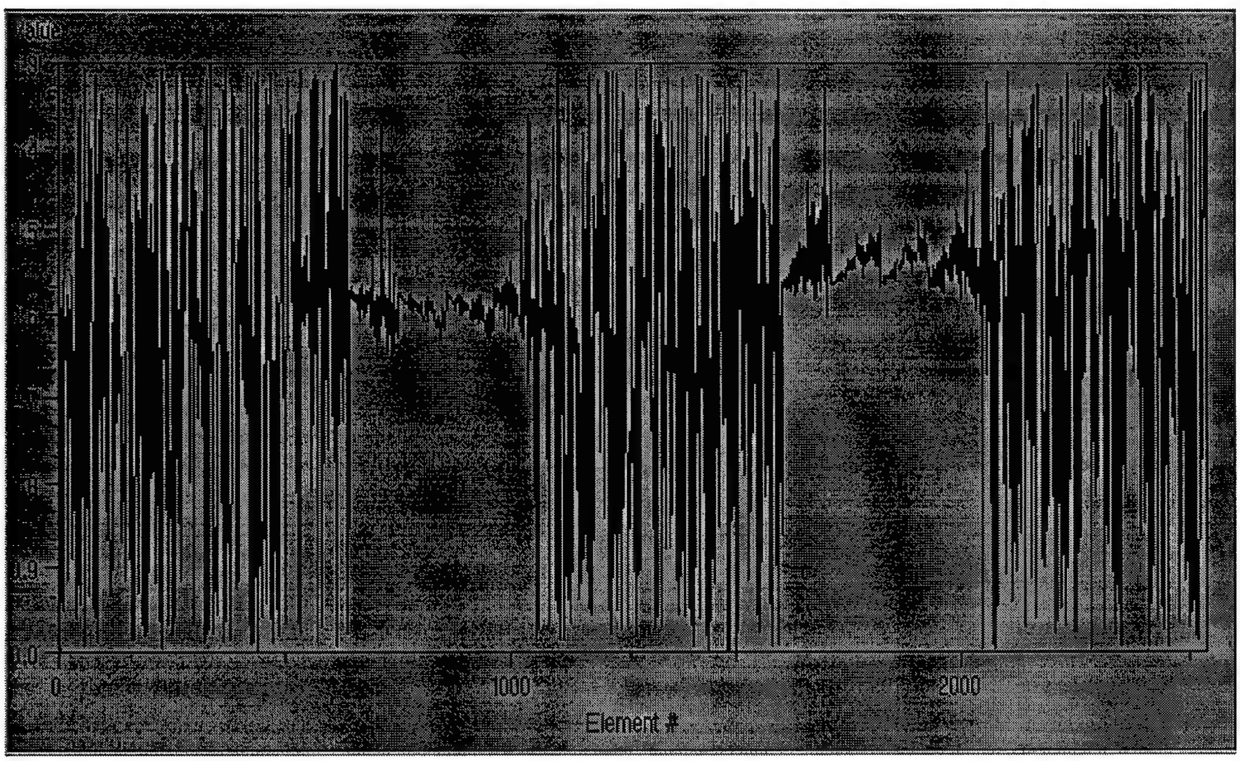


Figure 23. Data on Queue BRCC

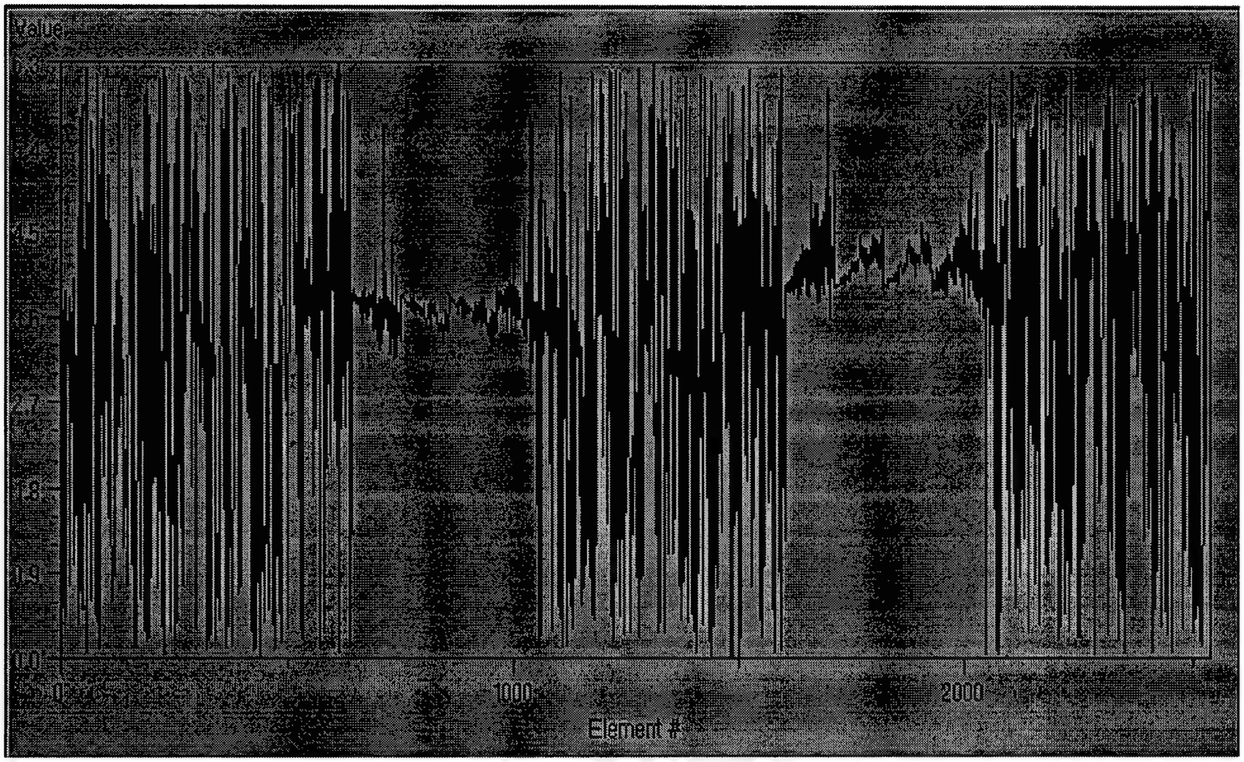


Figure 24. Data on Queue BRCN

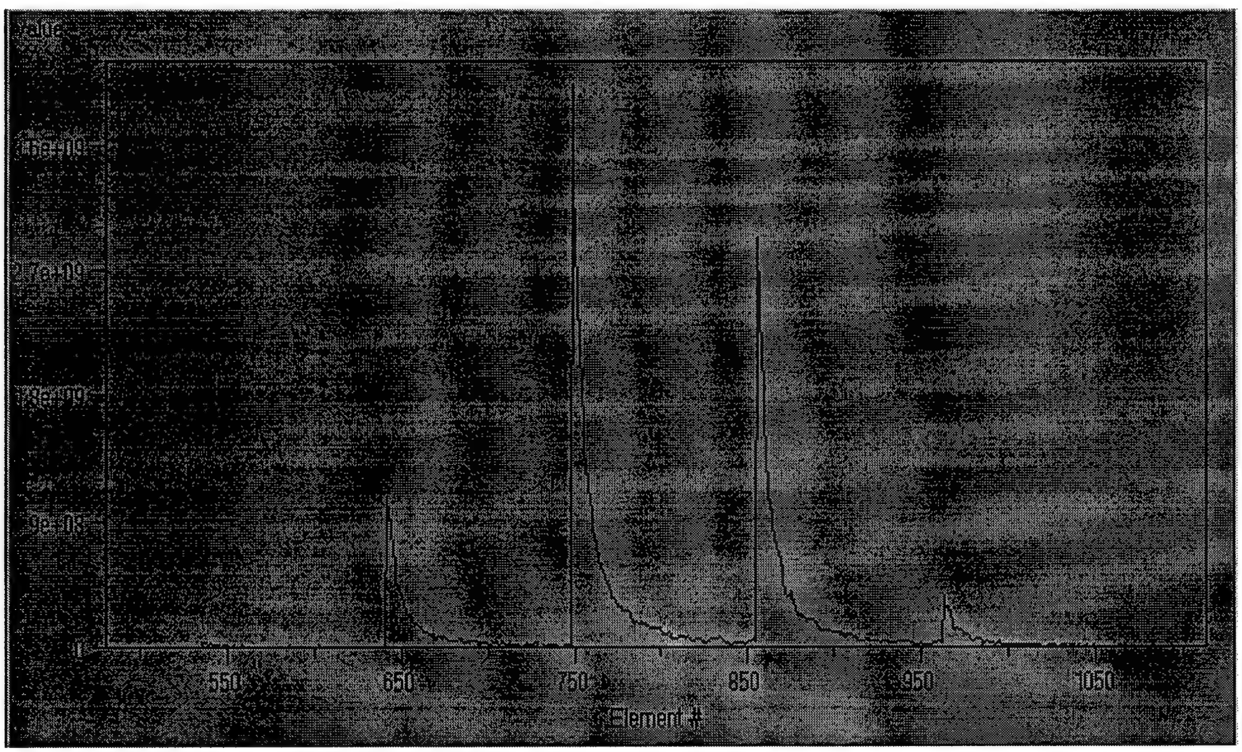


Figure 25. Data on Queue Nmean_FPSB

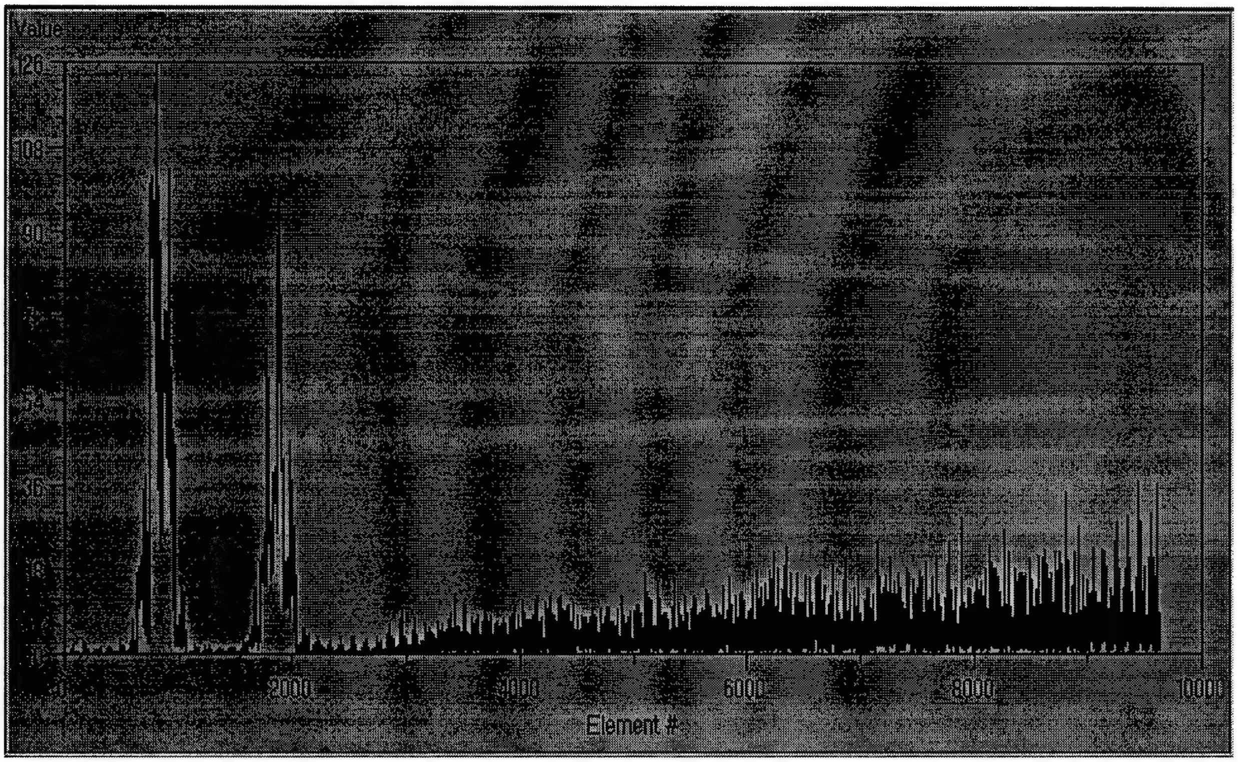


Figure 26. Data on Queue Nmean_NMWF

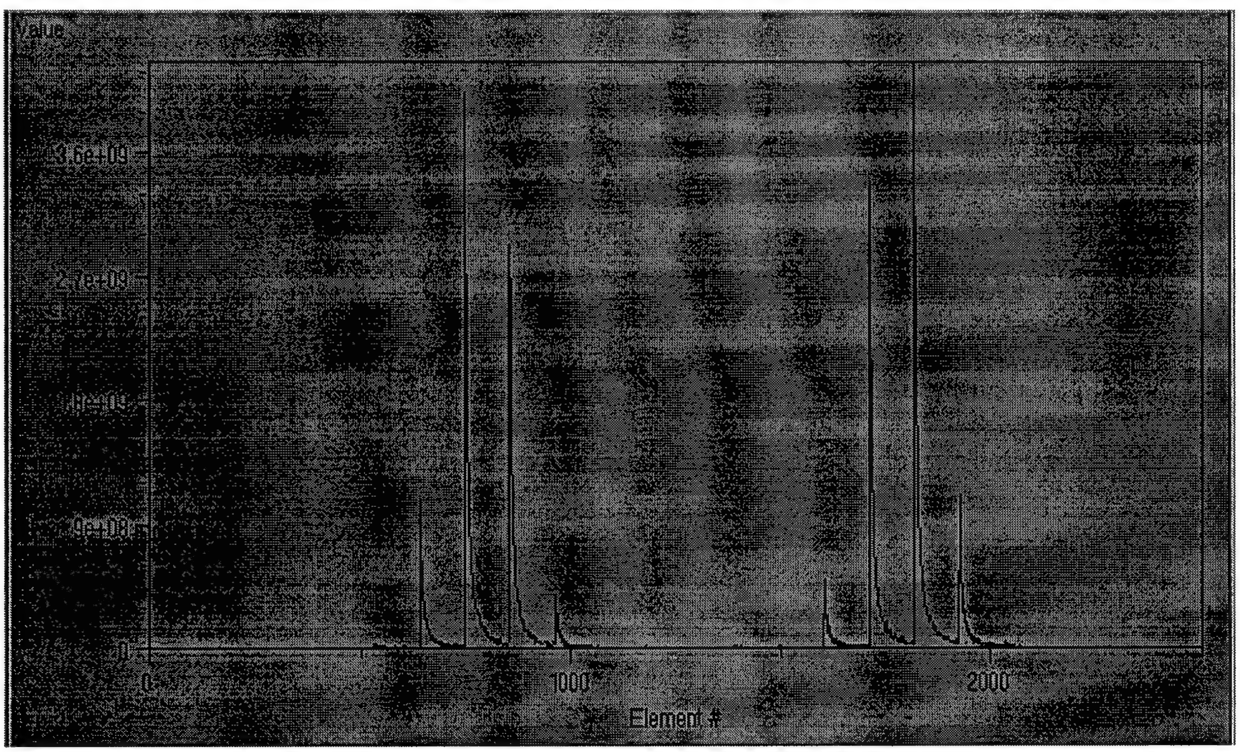


Figure 27. Data on Queue Nmean_UNWF

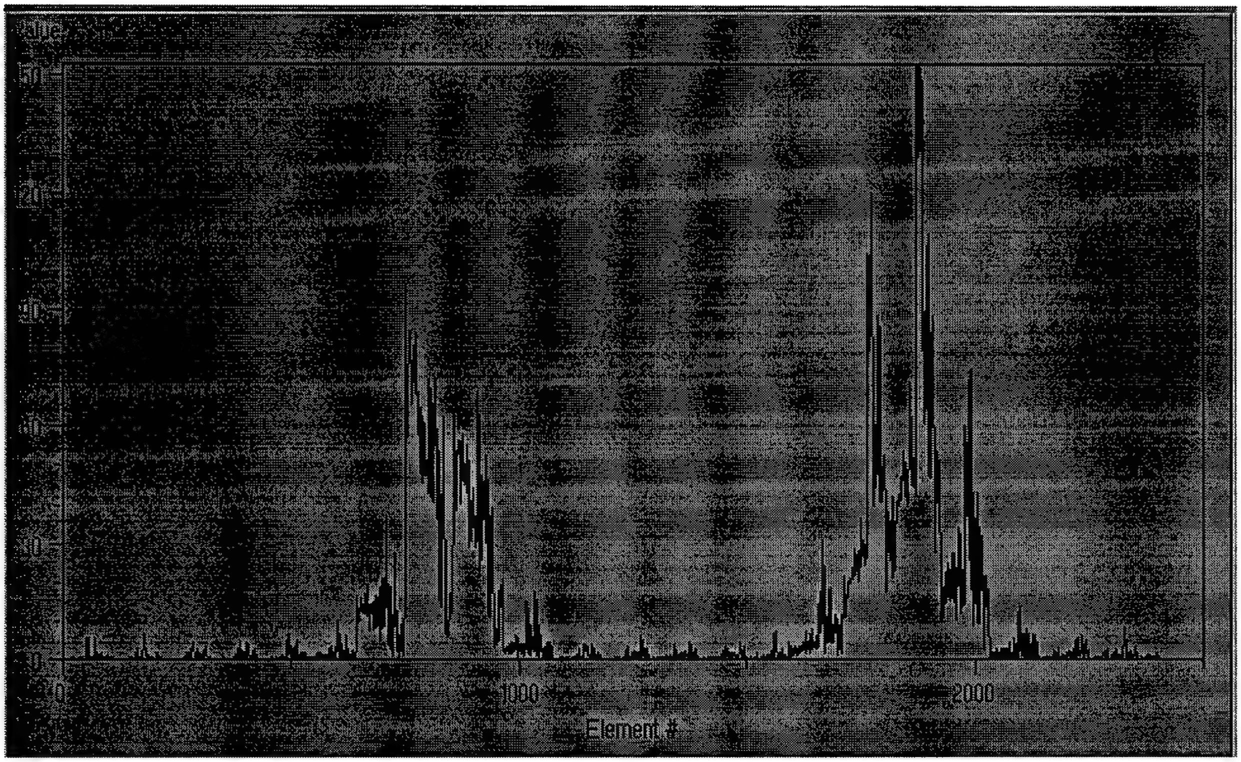


Figure 28. Data on Queue Cmean_NMWF

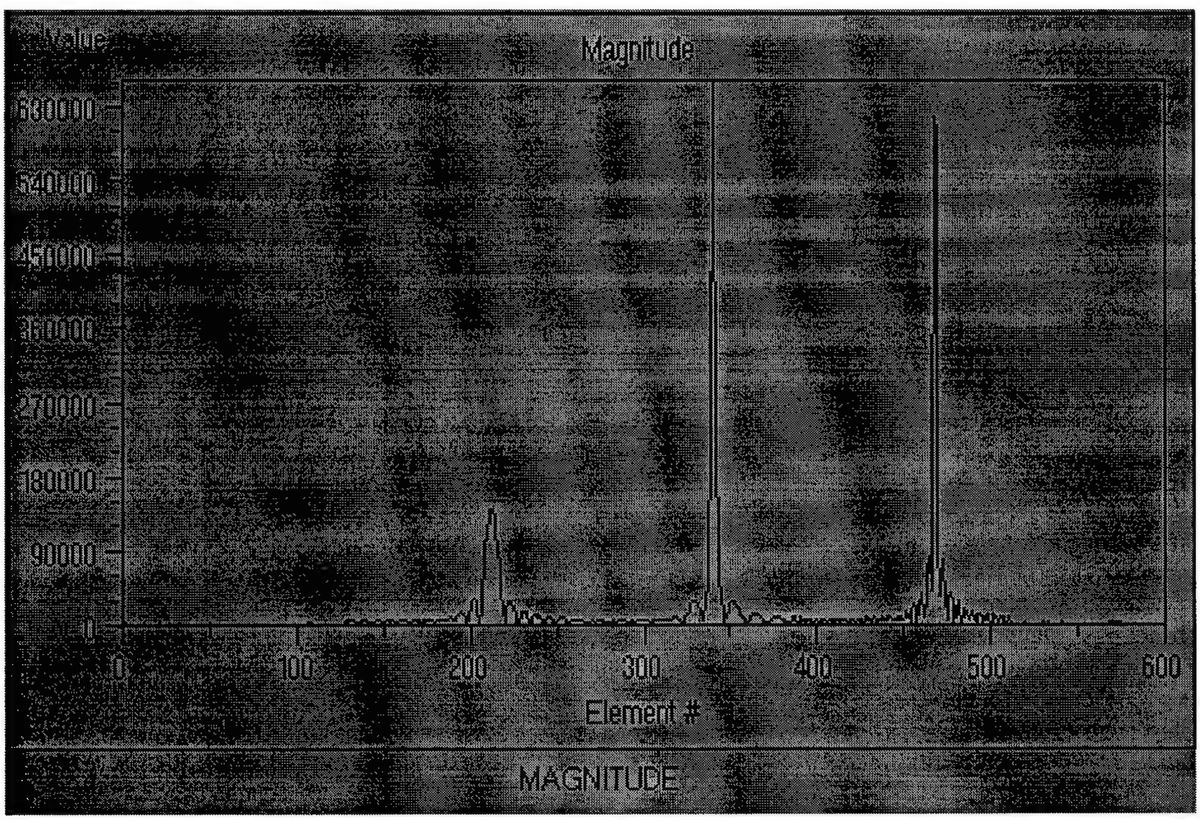


Figure 29. Data on Queue Audio_FLT

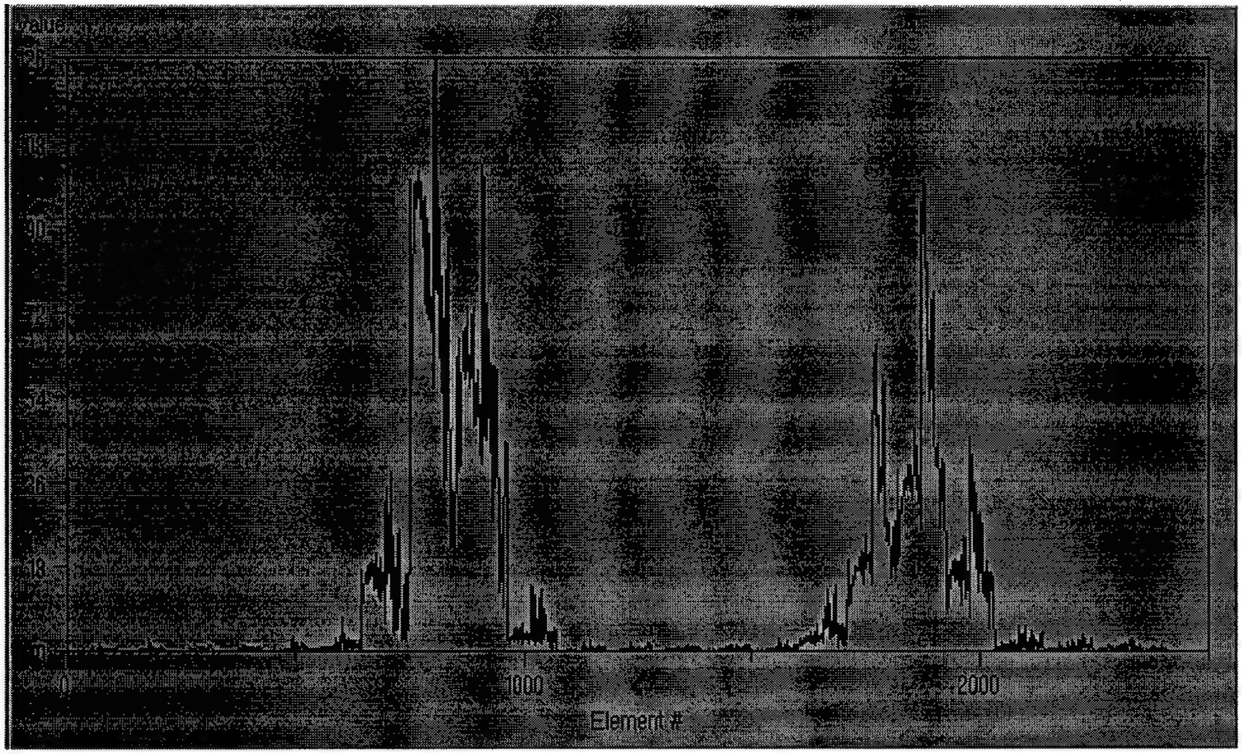


Figure 30. Data on Queue Waterfall_X10

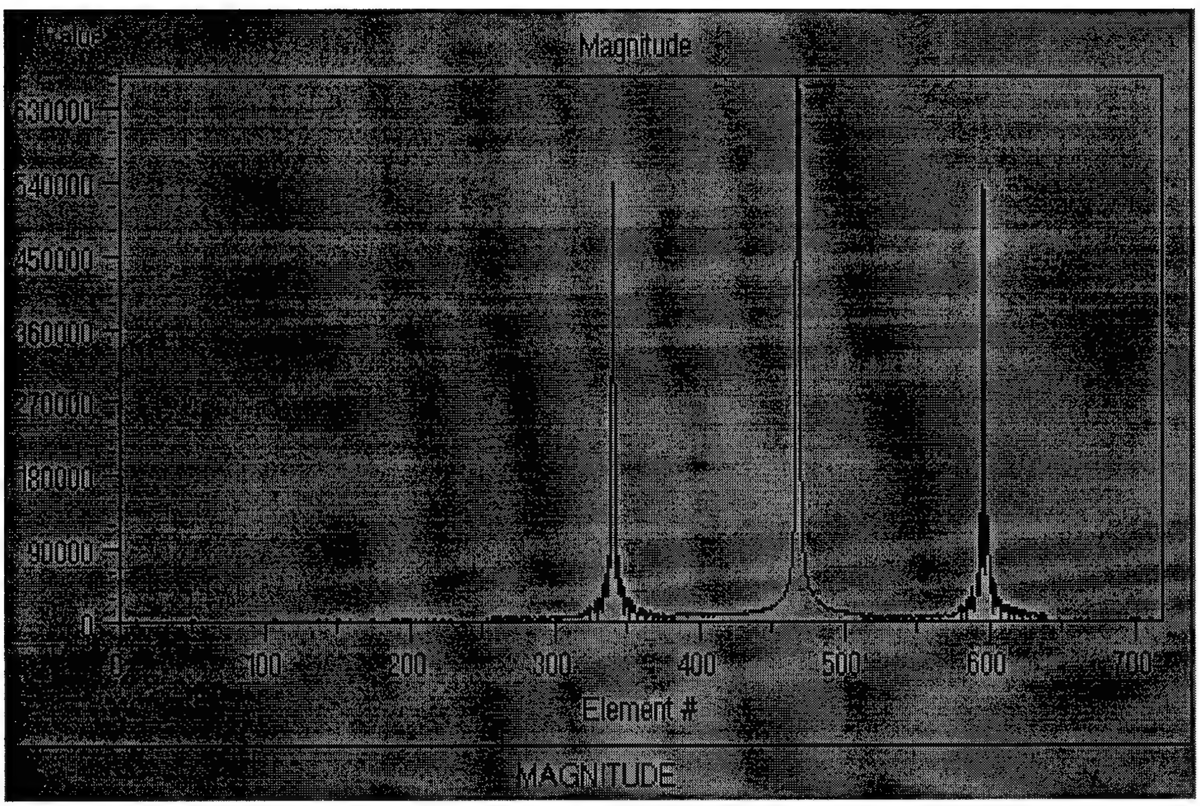


Figure 31. Data on Queue Ascan_X2

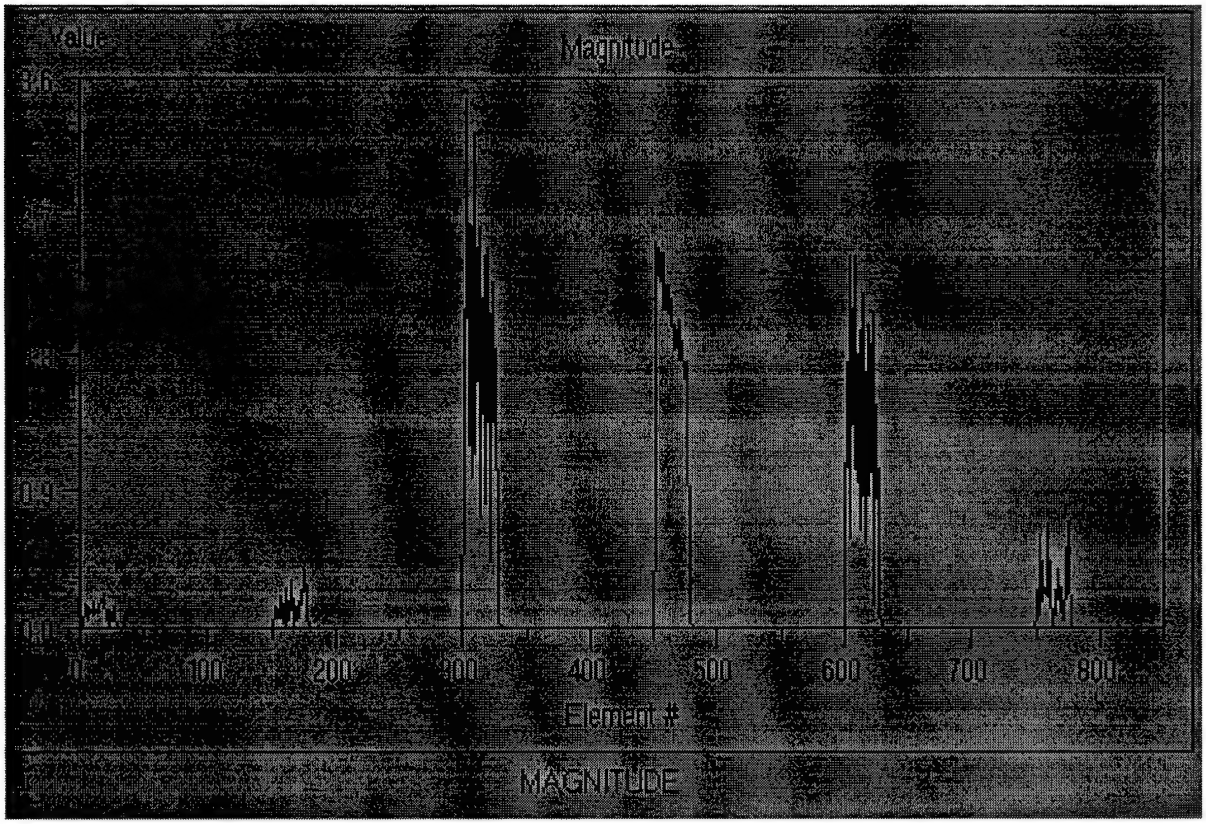


Figure 32. Data on Queue Ascan_X6

6.4.6 Graph Value Sets

The MCCI Autocoding Toolset performs partial instantiation of the graph at compile time. The values permissible for certain integer variables (GIPs and VARs) must be defined. Any parameter which affects the execution sequence and/or the memory map must be defined in the Graph Value Set. For the DICASS graph, the following parameters fall into this category:

PBINCF - total number of coarse and fine bins selected for output to AIU
LPB - CWL A-scan passband bin count
MPB - CWM A-scan passband bin count
SPB - CWS A-scan passband bin count

It is likely that these parameters could be eliminated from the Graph Value Set by modifying sections of the graph to use queues of mode v_array. In that case, only the maximum value of these parameters would have to be defined. This requires further investigation to determine the validity of this approach. Workarounds for several other parameters that initially were required to be in the Graph Value Set were found and implemented.

6.4.7 Status

Partition graphs were generated successfully for the converted DICASS graph and related subgraphs. All of the partitions were successfully autocoded with the following exceptions:

The Mode Change Synchronization Domain Primitive (D_MCS) was not implemented correctly due to a misinterpretation of the Q003 description. This resulted in an inability to use the implemented version as required by the application.

The Channel Gain Adjust (CGA) processing was originally implemented as a single subgraph, however each of the CW modes (CWL, CWM, and CWS) process a different data amount. This processing either needs to be converted to encompass queues of mode v_array, or else separate subgraphs need to be incorporated for each of the modes that process a different data amount.

The Merge construct was implemented to process the same amount of data from each member of the family of input queues. Some of the instantiations of Merge require a different amount of data from each member. The Autocoding Toolset does not currently support this capability.

The processing of the data for the displays using the VPACK primitive was not correctly understood, and therefore the processing implemented using D_ VSCT is believed to be incorrect.

D_PACK packs four four bit words into a 32 bit word, leaving the higher order bits zeroed. This should be modified to pack 8 four bit words into the 32 bit word.

In generating the FM subreplicas, the starting bin for the output is selected via a run-time expression. The current implementation of the FFT Domain Primitive does not support this parameter as run-time variable. The FFT implementation needs to be modified.

It is estimated that correcting these problems will require about a 0.75 personmonth effort.

The requirements for the interface to the display were not reviewed, and therefore the processing likely does not output the correct number of words.

An Input/output Procedure to generate simulated NS, EW and Omni signals was constructed. Other Input/Output procedures were not implemented. No Command Program was implemented.

6.4.8 Level of Effort

The level of effort required to perform each of the major functions associated with the conversion of the DICASS graph is shown in Table 2.

Task	Hours
Domain Primitives	387.5
Convert Graph	33.0
Convert Chains	22.0
Partition/Autocode	17.0
Test Partitions	87.5
Total	**547**

Table 2. Level of Effort

6.4.9 Conclusions and Recommendations

The DICASS graph and related subgraphs were readily converted to Domain Primitive Application Graphs. New Domain Primitives were implemented to encompass the functionality required by DICASS that was not in the existing Domain Primitive set. These new primitives are sonobuoy processing or display related.

A limitation of the current toolset is that there is no easy way to iconically designate partitions, and further the viewing of graph - subgraph connections is not available within DSPGRAPH. Each graph can be displayed individually, but simultaneous viewing of several graphs becomes cumbersome for large graphs. Little effort was made to create partitions that contained segments from different subgraphs. The number of partitions could be decreased by this method.

When viewing the Equivalent Application Graph, an additional limitation of DSPGRAPH is apparent. Partition Builder constructs node and queue names that are long. It does this in order to ensure uniqueness of names. A byproduct of this naming convention is that it is easy to trace back to the graph source of any entity. In order to read the names, the scale factor must be high; however, to view a large graph, the scale factor must be reduced.

Additionally, the automatic layout processing currently in DSPGRAPH is insufficient for graph with large number of nodes, and/or with many queues that create a "spiderweb."

The following actions are recommended to complete a meaningful demonstration of DICASS processing on a COTS platform:

1. Correct the deficiencies identified under the Section "Status" above.

2. The DICASS graph used for conversion is of unknown origin and seemed to contain some errors and modifications. The Merge construct was referenced with both integer queues and integer array queues, yet only integer queues are permitted by the PGM specification. Some queues were not attached at the head, others were not attached at the tail and contained no initialization data. Therefore, before proceeding,

a known correct version of the graph should be obtained, and used as the baseline. The modifications made should be implemented in this baseline.

3. Requirements for the display interfaced need to be reviewed, so that a thorough understanding of the interface can be obtained.

4. The Command Program interface needs to be reviewed and representative values for parameters identified, such that representative data sets can be generated.

5. Test vectors are required to test the converted graph. These data sets need to reflect the parameter values of item 4.

6. The converted graph should be embedded into a system that contains the display so that proper operation can be observed.

7. ILS Strategy

The architecture family MCCI proposes will support development of a COTS friendly ILS strategy. A board replacement maintenance strategy will be a fact of life for the majority of military system lifetimes. Replacement boards will use new technology and perhaps a new architecture. MCCI's architecture family concept will facilitate the introduction of new technology replacement boards without the attendant major software rewrites that would otherwise be necessary. Application reuse in hybrid new and old technology systems will be supported. An ILS strategy that integrates life cycle software support with board replacement logistics support will be supportable.

7.1 Board Replacement ILS Strategy

The architecture family will support a board replacement maintenance strategy. Introduction of a new technology generation into a vendor's product line usually involves an operating system upgrade. Operating system upgrades may support older technology generations for some period; e.g., Mercury Computer Systems, Inc.'s MCOS support of i860, Power PC, and SHARC based boards. The Autocoding Toolset's middleware level interfaces to operating systems will make any dependencies on operating system upgrades transparent to the maintainers. New technology boards may replace older boards without expensive software rewrite. It may be necessary, or expedient, to reassign tasks and threads to accommodate or take advantage of the new technology. This can be readily accomplished by re-autocoding the application with different partitioning directives. This is analogous to a recompile.

7.2 Board and Vendor Migration

At some point in airborne system life cycle, vendor support for older technology boards will inevitably be dropped. At that point, replacement of older, non-supported boards will be required. MCCI will continue to support boards as long as they are fielded. Transitioning board sets to new technology boards may be made a part of regularly scheduled major maintenance actions. Board replacement within older supported sets will serve maintenance needs in the interim. Again, no major application software

rewrite will be necessary to transition to new technology boards. All target specific modifications are encapsulated within the Autocoding Toolset and the corresponding run-time services. Transition to new technology board sets will also provide a natural opportunity to change board vendors. It is entirely possible that boards from multiple vendors will be used within the fleet. At the time a board set is replaced, it may be convenient to replace it with a set from an alternative vendor. Again, this may be undertaken without expensive software rewrite.

7.3 Incorporation of Performance Upgrades with Board Replacement

We have emphasized the minimal impact of board and vendor upgrades on software; however, it may be advantageous to use the additional capacity new technology boards will likely provide by introducing processing upgrades. Existing applications may be incrementally upgraded without major disruption of existing code. Additional channels may be added, new processes introduced, etc. These modifications can be easily integrated into the existing application. Repartitioning may be accomplished to best utilize the increased capacity without change to the existing application DPAGs. New application configurations with upgrades incorporated may then be provided in support of opportunistic or scheduled maintenance actions. Incremental upgrade of processing capacity may be made an ongoing activity and integrated into maintenance support to minimize impact on system availability.

Appendix A - Description of Chain CHN_ASNP

Portable Reusable Application Software
SBIR Phase I Final Report

October 28, 1998

Prepared by:
Management Communciations and Control, Inc. (MCCI)
2000 North Fourteenth Street, Suite 220
Arlington, VA 22201
Under Contract N68335-98-C-0140

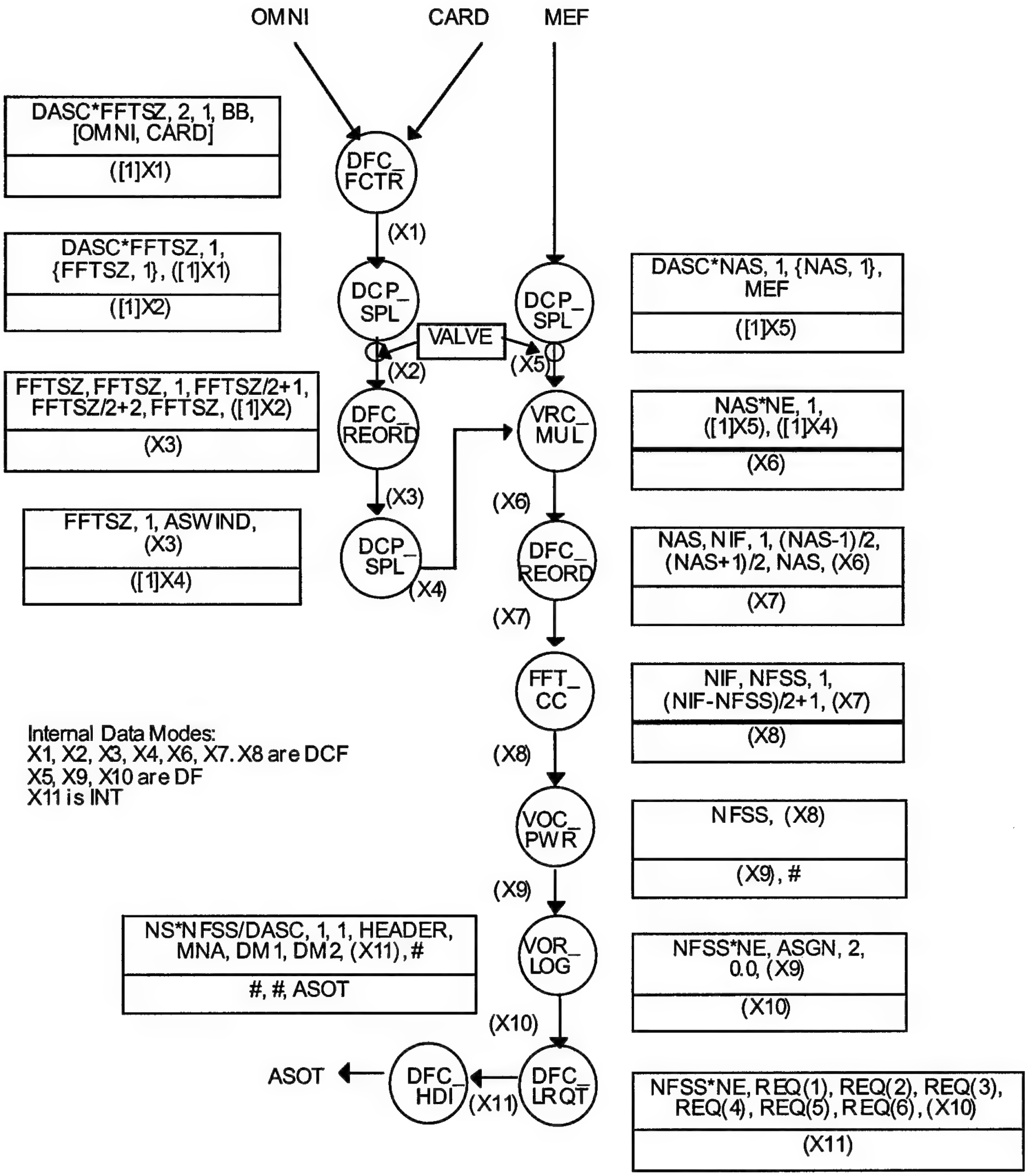
A-SCAN PROCESSING - CHN_ASNP

DESCRIPTION:

This chain performs the A-Scan time series processing for DICASS CW ping type data. OMNI and CARD contain the FFT input data. MEF contains the normalized coefficients for weighting the FFT data. Either OMNI or CARD is selected for processing; if BB is 1, OMNI is selected, otherwise CARD is selected. The selected FFT data are decimated and rearranged in frequency order. A sub-band of the reordered FFT data is then selected for further processing. After the selected FFT bins are weighted by the normalized weights, the weighted FFT data is inverse-transformed back into a complex time series. The complex time series is then square-law detected. The logarithm of the detected data is calculated and requantized into 8-bit data. ASOT is the output data containing the requantized data with a header inserted at the beginning. If VALVE = 0 the input data is consumed, but no additional processing is done, and ASOT is not output.

A-SCAN PROCESSING - CHN_ASNP (continued)

CHAIN TOPOLOGY:



A-SCAN PROCESSING - CHN_ASNP (continued)

ALGORITHM:

For the processing performed, see the algorithms in CDRL Q003 for DCP_SPL, DFC_FCTR, DFC_REORD, VRC_MUL, FFT_CC, VOC_PWR, VOR_LOG, DFC_LRQT, and DFC_HDI.

PARAMETER LIST:

PRIMITIVE = CHN_ASNP
PRIM_IN = DASC, NAS, NS, FFTSZ, NIF, NFSS, BB, VALVE, ASWIND, ASGN, REQ, HEADER, MNA, DM1, DM2, MEF, OMNI, CARD
PRIM_OUT = ASOT

MNEMONIC	INPUT DESCRIPTION	INPUT AMOUNT	MODE	RANGE
DASC	Decimation Rate	1	I	1 to 10
NAS	Sub-band size	1	I	4 to NIF
NS	Scans per Processing Block	1	I	1 to 32
FFTSZ	FFT Size	1	I	Constrained
NIF	IFFT Size	1	I	Constrained
NFSS	Selected IFFT Output Bins	1	I	1 to NIF
BB	Flow Control array	1	I	1 or 2
VALVE	Decimation Control Valve	1	I	0 or 1
ASWIND*	Band Selection Array	1	I Array(2)	•Constrained
ASGN	Amplitude Adjustment	1	DF	±mfloat
REQ*	Requantization Parameters	1	DF Array(6)	±mfloat
HEADER*	AIU Header	1	I Array(8)	•-32768 to 32767
MNA	AUI Header ORed Words	1	I Array(2)	•1 to 8
DM1	Data Mask One	1	I	-32768 to 32767
DM2	Data Mask Two	1	I	-32768 to 32767
MEF‡	Normalized Weights	DASC * NAS	DF	±mfloat
OMNI‡	Omni Data	DASC * FFTSZ	DCF	±mfloat
CARD‡	Cardioid Data	DASC * FFTSZ	DCF	±mfloat

‡ These ports govern multiple execution, NE = NS/DASC.
* Array slicing for ASWIND: S([i1,][j1,][K]) -> ASWIND(i).
* Array slicing for REQ: S([i1,][j1,][K]) -> REQ(i).
* Array slicing for HEADER: S([i1,][j1,][K]) -> HEADER(i).

A-SCAN PROCESSING - CHN_ASNP (continued)

MNEMONIC	INPUT DESCRIPTION	INPUT AMOUNT	MODE	RANGE
ASOT	A-Scan Output	1 or 0	I V_ARRAY (KK)	-32768 to 32767

Note: KK = (NS*NFSS)/DASC + 8

CONSTRAINTS:

- $1 \le$ ASWIND(2) $\le$ FFTSZ
- $1 \le$ ASWIND(2) + ASWIND(1) - 1 $\le$ FFTSZ
- ASWIND(1) = NAS

  FFTSZ = 2**k where k is an integer and $6 \le k \le 11$

  NIF = 2**k where k is an integer and $4 \le k \le 10$
- REQ(3) $\ge$ REQ(4)

  MOD(NS, DASC) = 0

  4NIF + NIF*(NS/DASC) $\le$ 16k - 1

  NS/DASC $\le$ 12

  READ(OMNI) = NS*FFTSZ

  READ(CARD) = NS*FFTSZ

  READ(MEF) = NS*NAS

  READ(OMNI) + OFFSET(OMNI) - CONSUME(OMNI) = 0

  READ(CARD) + OFFSET(CARD) - CONSUME(CARD) = 0
- KK $\le$ KMAX

Note: KMAX is the user-defined maximum size of the output V_ARRAY.

Appendix B - Generalized Mapping of Q003 Primitives to Domain Primitives

Portable Reusable Application Software SBIR Phase I Final Report

October 28, 1998

Prepared by:
Management Communications and Control Inc. (MCCI)
2000 North Fourteenth Street, Suite 220
Arlington, VA 22201
Under Contract N68335-98-C-0140

Generalized Mapping of
Q003 Primitives to Domain Primitives

The Q003 primitives are taken from CDRL Q003 December 1, 1993.

Note that Domain Primitives do not support fixed types.

The mapping is organized by Q003 name. Separate charts are used for each of the Q003 classifications (BFR_, FFT_, VCC_, etc.)

BFR_

Q003	DP	Comments
BFR_FREQ	D_BFRF	Family of weights has become array.
BFR_MFRQ	D_BFRF	Seems to be extended case of BFR_FREQ.
BFR_REL	---	
BFR_TRUE	---	

CDM_

Q003	DP	Comments
CDM_CFF	D_CDMF	Without multiplexing.
CDM_CVF	D_CDMV	Without multiplexing.
CDM_CVFM	D_CDMV	
CDM_MRVF	D_CDMV	With multiplexing.
CDM_RFF	D_CDMF	Without multiplexing.
CDM_RFFM	D_CDMF	With multiplexing.
CDM_RFIR	D_CDMFIR	Without multiplexing.
CDM_RVF	D_CDMV	Without multiplexing.

DCP_

Q003	DP	Comments
DCP_AVG1	D_AVG1	
DCP_AVGN	D_AVGN	DP not fully implemented.
DCP_CGA	D_CGA	
DCP_CGA1	---	
DCP_CLS	D_CLS	
DCP_CRB	D_CRB	
DCP_CSMG	---	
DCP_DEC	D_DEC	
DCP_EAVN	D_EAVN	DP not fully implemented.
DCP_ECLS	---	
DCP_FRQW	D_FRQW	
DCP_FRQWC	D_FRQWC	

DCP_ (continued)

Q003	DP	Comments
DCP_HAMN	D_HAMN	
DCP_INTD	D_INTD	
DCP_ISDR	---	
DCP_LAGI	D_LAGI	
DCP_LINT	D_LINT	
DCP_MEF	D_MEF	
DCP_MET	D_MET	
DCP_METD	---	
DCP_MWAG	D_MWAG	
DCP_MWGT	D_MWGT	
DCP_NME	D_NME	
DCP_NMED	---	
DCP_NORM	---	
DCP_NORM3	D_NORM3	Mean.
DCP_NSE	---	
DCP_RINT	D_RINT	
DCP_SMERGE	D_SMERGE	Family of sizes has become array.
DCP_SPL	D_SPL	
DCP_STI	D_STI	
DCP_TSS	D_TSS	
DCP_VDI	D_VDI	
DCP_VFILL	D_VFILL	

DFC_

Q003	DP	Comments
DFC_BMAX	---	
DFC_CAP	---	
DFC_CAT	D_CAT	Family of sizes has become array.
DFC_DMUX	D_DMUX	
DFC_DSCC	---	
DFC_DSD	D_DSD	
DFC_ERUP	---	
DFC_FCTR	D_FLOC	
DFC_FLOC	D_FLOC	
DFC_FMTPK	---	
DFC_HDI	D_HDI	V_array to vector and vice versa may be accomplished by D_VFILL.
DFC_IOVR	---	
DFC_LRQT	D_LRQT	
DFC_MCS	D_MCS	
DFC_MUX	D_MUX	
DFC_OTBD	---	
DFC_OTR	---	
DFC_PACK	D_PACK	
DFC_PSK	---	
DFC_REORD	D_REORD	
DFC_REP	D_REP	Equivalences output. D_REPNE Distinct output queues.

DFC_ (continued)

Q003	DP	Comments
DFC_REP2	D_REP D_REPNE	Equivalences output Distinct output queues.
DFC_REQ	---	
DFC_REQV	D_REQV	
DFC_SCAT	D_SCAT	
DFC_SEP	D_SEP	
DFC_STA	---	
DFC_SWTH	D_SWTH	?
DFC_TIME	---	
DFC_TSR	---	
DFC_UNPK6	---	
DFC_VCAT	D_CAT	Concatenation only.
DFC_VPAC	---	
DFC_VPC2	---	
DFC_VREP	D_REP D_REPNE	Equivalences output. Distinct output queues.
DFC_VSCT	D_VSCT	
DFC_VT	---	If maximum number of output elements desired is KMAX(Y), this can be done with D_REP or D_REPNE, but KK must be obtained elsewhere. If maximum number of output elements desired is less than KMAX(Y), D_REORD can be used.

DGP_

Q003	DP	Comments
DGP_BFWT	---	
DGP_CWFM	---	
DGP_HFMG	D_HFMG	
DGP_WWG	---	

DMC_

Q003	DP	Comments
DMC_CTOR	D_CTOR	
DMC_EMC	D_EMC	
DMC_FLIN	D_RTOI	
DMC_FXFL	---	

FFT_

Q003	DP	Comments
FFT_CC	D_FFT	
FFT_CC3	---	
FFT_CR	D_FFT	
FFT_R2C	---	
FFT_RC	D_FFT	

FIR_

Q003	DP	Comments
FIR_C1S	D_FIR1S	Without multiplexing.
FIR_C2S	D_FIR2S	Without multiplexing.
FIR_C7	---	
FIR_MC1S	D_FIR1S	With multiplexing.
FIR_MX23	---	
FIR_MX33	---	
FIR_MX7	---	
FIR_R19	---	
FIR_R1S	D_FIR1S	Without multiplexing.
FIR_R1SC	D_FIR1S	Without multiplexing.

IIR_

Q003	DP	Comments
IIR_C1S	D_IIR1S	Without multiplexing.
IIR_C22	---	

MOC_

Q003	DP	Comments
MOC_TPSE	D_MTRANS	

SSP_

Q003	DP	Comments
SSP_AGC	D_AGC	
SSP_BCOR	---	
SSP_BMS	--	-
SSP_CARD	D_CARD	
SSP_CCL	---	
SSP_COMV	---	
SSP_CVU	---	
SSP_DCD	D_DCD	
SSP_DNS	---	
SSP_DSC	---	
SSP_EST	---	
SSP_FBRG		
SSP_FPD	---	
SSP_GAG	---	
SSP_LPP	---	
SSP_LPP2	---	
SSP_LPPA	---	
SSP_LPPV		
SSP_MAP	---	
SSP_MBPP		
SSP_MEB	---	
SSP_PDF	---	
SSP_PPIN	---	
SSP_SYNO	D_SYNO	

SSP_ (continued)

Q003	DP	Comments
SSP_TDT	---	
SSP_TINT	---	
SSP_TRK	---	
SSP_UTD	---	
SSP_ZDT	D_ZDT	

VCC_

Q003	DP	Comments
VCC_VADD	D_VADD	
VCC_VDIV	D_VDIV	
VCC_VMUL	D_VMUL	
VCM_CTH2	D_CTH2	Parameter TN not used.
VCM_CTHS	D_CTH2	With TD = 0.
VCM_DIFM	D_DIFM	
VCM_DTH	D_DTH	
VCM_THCC	---	
VCM_THRS	---	
VCM_THRST	---	

VOC_

Q003	DP	Comments
VOC_CONJ	D_CONJ	
VOC_PWR	D_PWR	

VOR_

Q003	DP	Comments
VOR_ATN2	D_ATAN2	
VOR_IIND	D_INDX	Either Y or VY may be output. Easy change to let K be valid data size of B if Y is a v_array.
VOR_INDX	D_INDX	
VOR_LIN	D_LIN	
VOR_LOG	D_LOG	
VOR_MAG	D_MAG	
VOR_VACM	---	I SLIDE=0, D_VMUL may be used.
VOR_VCC2	D_VCC2	
VOR_VCIP		
VOR_VIND	D_INDX	Change to let K be valid data size of B.
VOR_VSQR	D_SQRT	
VOR_ZCC	D_ZCC	

VRC_

Q003	DP	Comments
VRC_MUL	D_VMUL	

VRR_

Q003	DP	Comments
VRR_GSUB	---	
VRR_INP	D_VINP	
VRR_VADD	D_VADD	
VRR_VDIV	D_VDIV	
VRR_VMUL	D_VMUL	
VRR_VSUB	D_VSUB	

Appendix C - Mapping of Parameters Q003 Primitives to Domain Primitives

Portable Reusable Application Software SBIR Phase I Final Report

October 28, 1998

Prepared by:
Management Communications and Control Inc. (MCCI)
2000 North Fourteenth Street, Suite 220
Arlington, VA 22201
Under Contract N68335-98-C-0140

Mapping of Parameters
Q003 Primitives to Domain Primitives

The following tables detail the correspondence between the parameters of a Q003 primitive call and the parameters of the corresponding Domain Primitive call(s).

Domain Primitives that are marked "vp" produce different amounts of output under different circumstances. Therefore they must be output nodes in a Partition Graph.

BFR_FREQ => D_BFRF

DP param	Q003 param	Comments
D_BFRF		
NF	NF (1)	Must be GIP.
NC	NC (2)	Must be GIP.
NB	NB (3)	Must be GIP.
W	[1..NB]W (5)	Family of weights must be made into array (of arrays), or a vector if W is input as a queue. The order of the elements in W has also been changed: for a given W array corresponding to an output beam, the array has as many rows as there are input family members of X, and each row has as many columns as there areelements in an X vector.
X	X (4)	
Y	Y	

BFR_MFRQ => D_BFRF

DP param	Q003 param	Comments
NF	NF (1)	Must be GIP.
NC	NC (2)	Must be GIP.
NB	NB (3)	Must be GIP.
W	[1..NB]W (5)	Family of weights must be made into array (of arrays), or a vector if W is input as a queue. The order of the elements in W has also been changed: for a given W array corresponding to an output beam, the array has as many rows as there are input family members of X, and each row has as many columns as there are elements in an X vector.
X	X (4)	
Y	Y	

BFR_REL => ---

DP param	Q003 param	Comments

BFR_TRUE => ---

DP param	Q003 param	Comments

CDM_CFF => D_CDMF

DP param	Q003 param	Comments
N	N (1)	Must be GIP.
MX	1 or UNUSED	Must be GIP if used.
FG	FG (2)	
NC	NC (3)	Must be GIP.
I	I (4)	
NP	NP (6)	May be GIP if NP' is unused.
X	X (5)	
Y	Y	
NP'	NP'	

CDM_CVF => D_CDMV

DP param	Q003 param	Comments
N	N	
MX	1 or UNUSED	Must be GIP if used.
Flag	FG	
M	M	Must be GIP.
F	F	Must be GIP or VAR array of size 1.
FS	FS	Must be GIP or VAR.
NP	NP	May be GIP if NP' is unused.
X	X	
Y	Y	
NP'	NP'	

CDM_MRVF => D_CDMV

DP param	Q003 param	Comments
N	N	
MX	MX	Must be GIP.
Flag	FG	
M	M	Must be GIP.
F	F	Must be GIP or VAR array of size 1.
FS	FS	Must be GIP or VAR.
NP	NP	May be GIP if NP' is unused.
X	X	
Y	Y	
NP'	NP'	

CDM_RFF => D_CDMF

DP param	Q003 param	Comments
N	N (1)	Must be GIP.
MX	1 or UNUSED	Must be GIP if used.
FG	FG (2)	
NC	NC (3)	Must be GIP.
I	I (4)	
NP	NP (6)	May be GIP if NP' is unused.
X	X (5)	
Y	Y	
NP'	NP'	

CDM_RFFM => D_CDMF

DP param	Q003 param	Comments
N	N (1)	Must be GIP.
MX	MX (2)	
FG	FG (3)	
NC	NC (4)	Must be GIP.
I	I (5)	
NP	NP (7)	May be GIP if NP' is unused.
X	X (6)	
Y	Y	
NP'	NP'	

CDM_RFIR => D_CDMFIR

DP param	Q003 param	Comments
N	N	Must be GIP.
MX	1 or UNUSED	Must be GIP if used.
FG	FG	
NC	NC	Must be GIP.
I	I	
NP	NP	May be GIP if NP' is unused.
NT	NT	Must be GIP.
D	D	Must be GIP.
B	B	
X	X	
Y	Z	
NP'	NP'	

CDM_RVF => D_CDMV

DP param	Q003 param	Comments
N	N	
MX	1 or UNUSED	Must be GIP if used.
Flag	FG	
M	M	Must be GIP.
F	F	Must be GIP or VAR array of size 1.
FS	FS	Must be GIP or VAR.
NP	NP	May be GIP if NP' is unused.
X	X	
Y	Y	
NP'	NP'	

DCP_AVG1 => D_AVG1

DP param	Q003 param	Comments
N	N	Must be GIP.
Flag	FG	
X	X	
Y	Y	

DCP_AVGN => D_AVGN Waiting for implementation decision.

DP param	Q003 param	Comments
N	N	Must be GIP.
M	M	
Flag	FG	
K	K	
A	A	
X	X	
Y	Y	
K'	K'	
A'	A'	

DCP_CGA => ---

DP param	Q003 param	Comments
N	N (5)	Must be GIP.
PMAX	PMAX (1)	
SMAX	SMAX (2)	
REINIT	REINIT (3)	
RHO	RHO (4)	
BSI	BSI (6)	
BSS	BSSAVE (7)	
SCNT	SCNT (8)	
PCNT	PCNT (9)	
PW	PW (10)	Must be array of size 2.
OMNI	OMNI (11)	
NS	NS (12)	
EW	EW (13)	
SCNT'	SCNT'	
PCNT'	PCNT'	
PW'	PW'	Must be array of size 2.
BS	BS	
BSS'	BSS'	

DCP_CLS => D_CLS

DP param	Q003 param	Comments
PIC	PIC	
CDF	CDF	
C	C	
CRL	CRL	
CRH	CRH	
WIND	WIND	
R	R	
NC	NC	
SI	SI	
CLI	CLI	
CLF	CLF	
CBS	CBS	
PHS	PHS	
T	T	
BEAR	BEAR	
X	X	
VY	VY	
R′	R′	
NC′	NC′	
SI′	SI′	
CLI′	CLI′	
CLF′	CLF′	
CBS′	CBS′	

DCP_CRB => ---

DP param	Q003 param	Comments
N	N	
NDV	NDV	
RSL	RSL	
B	B	
KA	KA	
KB	KB	
Q	Q	
CBN	CBN	
Z	Z	
LBIN	LBIN	
DW	DW	
CB	CB	
DCB	DCB	
MT	MT	
CB′	CB′	
DCB′	DCB′	
CBOFF	CBOFF	
DCBOFF	DCBOFF	

DCP_CSMG => ---

DP param	Q003 param	Comments

DCP_DEC => D_DEC

DP param	Q003 param	Comments
N	N	Must be GIP.
D	D	Must be GIP.
X	X	
Y	Y	

DCP_EAVN => D_EAVN vp Not fully implemented.

DP param	Q003 param	Comments
N	N (2)	Must be GIP.
M	M (1)	
A	A (3)	
Flag	FG (4)	
Y0	Y0 (5)	
X	X (6)	
Y	Y	
Y0'	Y0'	

DCP_FRQW => D_FRQW

DP param	Q003 param	Comments
N	N	Must be GIP.
M	M	Must be GIP.
NW	NW	Must be GIP.
B	B	
TS	TS	Must be GIP.
W	W	
X	X	
Y	Y	

DCP_FRQWC => D_FRQWC

DP param	Q003 param	Comments
N	N	Must be GIP.
M	M	Must be GIP.
NW	NW	Must be GIP.
B	B	
TS	TS	Must be GIP.
W	W	
X	X	
Y	Y	
YC	YC	

DCP_HAMN => D_HAMN

DP param	Q003 param	Comments
N	N	Must be GIP.
Flag	FG	
X	X	
Y	Y	

DCP_INTD => D_INTD

DP param	Q003 param	Comments
N	N	Must be GIP.
MX	MX	Must be GIP.
NW	NW	Must be GIP.
M	M	Must be GIP.
INCR	INCR	
RY	RY	Must be GIP.
W	W	
X	X	
Y	Y	

DCP_ISDR => ---

DP param	Q003 param	Comments

DCP_LAGI => D_LAGI

DP param	Q003 param	Comments
NS	NS	Must be GIP.
B	B	Must be GIP.
CL	CL	Must be GIP.
V	V	Must be GIP.
W	W	Must be array.
STB	STB	Must be array.
X	X	
Y	Y	

DCP_LINT D_LINT

DP param	Q003 param	Comments
N	N	Must be GIP.
M		Must be GIP.
DX	DX	
X0	X0	
X	X	
Y	Y	
Z	Z	

DCP_MEF => D_MEF

DP param	Q003 param	Comments
NS	NS (1)	Must be GIP.
NB	NB (2)	Must be GIP.
WF	WF (3)	Must be GIP.
GF	GF (4)	Must be GIP.
KACF	KACF (5)	
KARF	KARF (6)	
NA	NA (7)	Must be GIP if used.
AW	AW (8)	
B	B (9)	
M	M (10)	Must be GIP or VAR
NY	NY (11)	Must be GIP.
SB	SB (12)	
FC	FC (13)	
CB0	CB0 (14)	
EPF	EPF (16)	
POF	POF (17)	
RPF	RPF (18)	
X	X (15)	
Y	Y	
SM	SM	

DCP_MET => D_MET vp

DP param	Q003 param	Comments
NS	NS (6)	Must be GIP.
NB	NB (1)	Must be GIP.
WF	WT (2)	Must be GIP.
GF	GT (3)	Must be GIP.
KACF	KACT (4)	
KARF	KART (5)	
SUMLT	SUMLT (7)	
C	C (8)	
R	R (9)	
CP	CP (10)	
Flag	BFLAG (11)	
EPF	EPT (13)	
POF	POT (14)	
RPF	RPT (15)	
X	X (12)	
SUMLT′	SUMLT′	
C′	C′	
R′	R′	
CP′	CP′	
MT	MT	

DCP_MWAG => D_MWAG

DP param	Q003 param	Comments
N	N	Must be GIP.
W	W	Must be GIP.
L	L	Must be GIP.
X	X	
Y	Y	

DCP_MWGT => D_MWGT

DP param	Q003 param	Comments
NX	NX	Must be GIP.
NV	NV	Must be GIP.
J	J	Must be GIP.
K	K	
W	W	
B	B	
II	INDX	
X	X	
Y	Y	

DCP_NME => D_NME

DP param	Q003 param	Comments
N	N	Must be GIP.
K	K	
L	L	
W	W	Must be GIP.
X	X	
Y	Y	

DCP_NMED => ---

DP param	Q003 param	Comments

DCP_NORM3 => D_NORM3

DP param	Q003 param	Comments
N	N	Must be GIP.
W	W	Must be GIP.
G	G	Must be GIP.
T	T	
WT	WT	
FG	UNUSED	
X	X	
NME	NME	
Y	Y	

DCP_NSE => ---

DP param	Q003 param	Comments

DCP_RINT => D_RINT

DP param	Q003 param	Comments
N	N	Must be GIP.
L	L	Must be GIP.
K	K	
SF	SF	Must be GIP.
Y0	Y0	
X0	X0	
X	X	
Y	Y	
Y0′	Y0′	
X0′	X0′	

DCP_SMERGE => D_SMERGE

DP param	Q003 param	Comments
NB	NB	Must be GIP.
N	[1..NB]N	Must be GIP. Family of sizes must be made into array.
NW	NW	Must be GIP.
W	W	
X	X	
Y	Y	

DCP_SPL => D_SPL

DP param	Q003 param	Comments
N	N	Must be GIP.
M	NB	Must be GIP.
B	BLS	Must be GIP.
X	X	
Y	Y	

DCP_STI => D_STI vp

DP param	Q003 param	Comments
N	N (1)	Must be GIP.
M	CL (3)	Must be GIP.
K	K (4)	
A	A (5)	
X	X (6)	
Y	Y (3)	
K′	K′ (1)	
A′	A′ (2)	

DCP_TSS => D_TSS vp

DP param	Q003 param	Comments
N	N (2)	Must be GIP.
C	COUNT (3)	
T	THRES (4)	Must be GIP.
F	F (5)	
S	S (6)	
X	X (7)	
C′	C′	
F′	F′	
VAR	VAR	
STD	STD	
MEAN	MEAN	

DCP_VDI => D_VDI vp

DP param	Q003 param	Comments
N	N (1)	Must be GIP.
M	CL (3)	Must be GIP.
V	V (4)	Must be GIP.
K	K (5)	
A	A (6)	
X	X (7)	
Y	Y (3)	
K′	K′ (1)	
A′	A′ (2)	

DCP_VFILL => D_VFILL

DP param	Q003 param	Comments
N	N	Must be GIP.
P	P	Must be GIP.
J	J	
V	V	
X	X	
Y	Y	

DFC_BMAX => ---

DP param	Q003 param	Comments

DFC_CAP => ---

DP param	Q003 param	Comments

DFC_CAT => D_CAT

DP param	Q003 param	Comments
M	M	Must be GIP.
NC	NC	Must be GIP.
N	[1..NB]N	Must be GIP. Family of sizes must be made into array.
X	X	
Y	Y	

DFC_DMUX => D_DMUX

DP param	Q003 param	Comments
N	N	Must be GIP.
M	M	Must be GIP.
X	X	
Y	Y	

DFC_DSD => D_DSD

DP param	Q003 param	Comments
N	N	Must be GIP.
T	T	Must be GIP or VAR.
C1	C1	Must be GIP or VAR.
C2	C2	Must be GIP or VAR.
C3	C3	Must be GIP or VAR.
M	M	
S	S	
X	X	
Y	Y	

DFC_ERUP => ---

DP param	Q003 param	Comments

DFC_FCTR => D_FLOC vp

DP param	Q003 param	Comments
K	K	Must be GIP array of size N, each element set to value of K.
N	N	Must be GIP.
M	M	Must be GIP.
B	B	
X	X	
Y	Y	

DFC_FLOC => D_FLOC vp

DP param	Q003 param	Comments
K	K	Must be GIP array of size N, each element set to value of K.
N	N	Must be GIP.
M	M	Must be GIP.
B	B	
X	X	
Y	Y	

DFC_FMTPK => ---

DP param	Q003 param	Comments

DFC_HDI => D_HDI

DP param	Q003 param	Comments
N	N (1)	Must be GIP.
FH	FH (3)	
MN	MN (4)	Must be array.
DM1	DM1 (5)	
DM2	DM2 (6)	
X	X/VX (7)	
M	M (8/9)	
Y	Y	
VY	VY	

DFC_IOVR => ---

DP param	Q003 param	Comments

DFC_LRQT => D_LRQT

DP param	Q003 param	Comments
N	N (1)	Must be GIP.
C	C (2)	
O	O (3)	
UR	UR (6)	
LR	LR (7)	
UL	UC (4)	
LL	LC (5)	
X	X (8)	
Y	Y	

DFC_MCS => ---

DP param	Q003 param	Comments
FC	FC	
K	K	Must be GIP.
N	N	Must be GIP.
NAB	NAB	
M	M	Must be GIP ARRAY(5).
MM	MM	Must be GIP.
W	W	Must be GIP ARRAY(5).
B	B	
CC	CC	
C	C	
CX	CX	
X	X	
FCS'	FCS'	
C'	C'	
X'	X'	
Y1	Y1	
Y2	Y2	
Y3	Y3	
Y4	Y4	
Y5	Y5	

DFC_MUX => D_MUX

DP param	Q003 param	Comments
N	N	Must be GIP.
M	M	Must be GIP.
X	X	
Y	Y	

DFC_OTBD => ---

DP param	Q003 param	Comments

DFC_OTR => ---

DP param	Q003 param	Comments

DFC_PACK => D_PACK

DP param	Q003 param	Comments
NX	NX	Must be GIP.
NY	NY	Must be GIP.
M	M	Must be GIP.
B	B	Must be GIP.
FG	FG	
RV0	RV0	
X	X	
Y	Y	

DFC_PSK => ---

DP param	Q003 param	Comments

DFC_REORD => D_REORD

DP param	Q003 param	Comments
N	N	Must be GIP.
M	M	Must be GIP.
A	A	
B	B	
C	J	
D	K	
X	X	
Y	Y	

DFC_REP => D_REP (Equivalenced output queues. Output queues cannot be initialized.)

DP param	Q003 param	Comments
N	N	Must be GIP.
M	M	Must be GIP.
X	X	
Y	Y	

=> D_REPNE vp Distinct output queues.

DP param	Q003 param	Comments
N	N	Must be GIP.
M	M	Must be GIP.
S	UNUSED	
X	X	
Y	Y	

DFC_REP2 => D_REP (Equivalenced output queues. Output queues cannot be initialized.)

DP param	Q003 param	Comments
N	N	Must be GIP.
M	M	Must be GIP.
X	X	
Y	Y	

=> D_REPNE vp Distinct output queues.

DP param	Q003 param	Comments
N	N	Must be GIP.
M	M	Must be GIP.
S	UNUSED	
X	X	
Y	Y	

DFC_REQ => D_REQ

DP param	Q003 param	Comments
N	N	Must be GIP.
C	C	Must be array.
X	X	
Y	Y	
Z	Z	

DFC_REQV => D_REQV

DP param	Q003 param	Comments
N	N	Must be GIP.
A	A	
FD	FD	
NL	NL	Must be GIP.
KI	KI	
X	X	
Y	Y	
Z	Z	

DFC_SCAT => D_SCAT vp

DP param	Q003 param	Comments
NC	NC	Must be GIP.
C	C	
M	M	Must be GIP.
N	N	Must be GIP.
X	X	
Y	Y	
UY	UY	
K	K	

DFC_SEP => D_SEP

DP param	Q003 param	Comments
N	N	Must be GIP.
M	M	Must be GIP.
X	X	
Y	Y	

DFC_STA => ---

DP param	Q003 param	Comments

DFC_SWTH **D_SWTH** vp DFC_SWTH works in conjunction with the Q003 function MERGE. D_SWTH is a standalone primitive. Hence the following parameter associations are merely guidelines; the parameter C may need to be altered to allow D_SWTH to perform like DFC_SWTH.

DP param	Q003 param	Comments
N	ream(X)	Must be GIP.
M	members(Y)	Must be GIP.
C	C	See note above.
X	X	
Y	Y	

DFC_TIME => ---

DP param	Q003 param	Comments

DFC_TSR => ---

DP param	Q003 param	Comments

DFC_UNPK6 => ---

DP param	Q003 param	Comments

DFC_VCAT => D_CAT Concatenation only, with NC = 1. If NC > 1, each input family member must previously be run through D_CAT to combine each set of v_arrays into one v_array. For reordering, a combination of D_SEP and D_CAT may be used.

DP param	Q003 param	Comments
M	M	Must be GIP.
NC	NC	Must be GIP.
N	UNUSED	
X	X	
Y	Y	

DFC_VPAC => ---

DP param	Q003 param	Comments

DFC_VREP => D_REP Equivalenced output queues. Output queues cannot be initialized.

DP param	Q003 param	Comments
N	UNUSED	
M	M	Must be GIP.
X	X	
Y	Y	

=> D_REPNE vp Distinct output queues.

DP param	Q003 param	Comments
N	UNUSED	
M	M	Must be GIP.
S	UNUSED	
X	X	
Y	Y	

DFC_VSCT => D_VSCT

DP param	Q003 param	Comments
NC	NC	Must be GIP.
NY	NY	Must be GIP.
CM	CM	
CNC	CNC	
FG	FG	
F0	F0	
M	M	Must be GIP.
X	X	
VY	VY	
KK	KK	

DFC_VT => ---

If the maximum number of output elements desired is KMAX(Y), DFC_VT may be accomplished with D_REP or D_REPNE. If the maximum number of output elements desired is less than KMAX(Y), D_REORD may be used. KK is the amount of valid data in each output v_array, but no Domain primitive exists to extract this information from a queue of v_arrays.

DGP_BFWT => ---

DP param	Q003 param	Comments

DGP_CWFM => ---

DP param	Q003 param	Comments

DGP_HFMG => ---

DP param	Q003 param	Comments
NY	NY	Must be GIP.
A	A	
DC	DC	
FOTC	FOTC	
P0	P0	
TCFS	TCFS	
I	I	
Y	Y	
I'	I'	

DGP_WWG => ---

DP param	Q003 param	Comments

DMC_CTOR => D_CTOR

DP param	Q003 param	Comments
N	N	Must be GIP.
X	Z	
Y	X	
Z	Y	

DMC_EMC => D_EMC

DP param	Q003 param	Comments
N	N	Must be GIP.
X	X	
Y	Y	

DMC_FLIN => D_RTOI

DP param	Q003 param	Comments
N	N	Must be GIP.
A	A	
B	B	
X	X	
Y	Y	

DMC_FXFL ---

DP param	Q003 param	Comments

FFT_CC => D_FFT

DP param	Q003 param	Comments
N	N	Must be GIP.
M	M	Must be GIP.
FI	FI	Must be GIP.
B	B	Must be GIP.
Ov	UNUSED	
X	X	
Y	Y	

FFT_CC3 => ---

DP param	Q003 param	Comments

FFT_CR => D_FFT

DP param	Q003 param	Comments
N	N	Must be GIP.
M	M	Must be GIP.
FI	FI	Must be GIP.
B	B	Must be GIP.
Ov	UNUSED	
X	X	
Y	Y	

FFT_R2C => ---

DP param	Q003 param	Comments

FFT_RC => D_FFT

DP param	Q003 param	Comments
N	N	Must be GIP.
M	M	Must be GIP.
FI	FI	Must be GIP.
B	B	Must be GIP.
Ov	UNUSED	
X	X	
Y	Y	

FIR_C1S => D_FIR1S

DP param	Q003 param	Comments
N	N	Must be GIP.
MX	1 or UNUSED	Must be GIP if used.
NT	NT	Must be GIP.
D	D	Must be GIP.
A	A	
X	X	
Y	Y	

FIR_C2S => D_FIR2S

DP param	Q003 param	Comments
N	N	Must be GIP.
MX	1 or UNUSED	Must be GIP if used.
NT1	NT1	Must be GIP.
NT2	NT2	Must be GIP.
D1	D1	Must be GIP.
D2	D2	Must be GIP.
A	A	
X	X	
Y	Y	

FIR_C7 => ---

DP param	Q003 param	Comments

FIR_MC1S => D_FIR1S

DP param	Q003 param	Comments
N	N	Must be GIP.
MX	MX	Must be GIP.
NT	NT	Must be GIP.
D	D	Must be GIP.
A	A	
X	X	
Y	Y	

FIR_MX23 => ---

DP param	Q003 param	Comments

FIR_MX33 => ---

DP param	Q003 param	Comments

FIR_MX7 => ---

DP param	Q003 param	Comments

FIR_R19 => ---

DP param	Q003 param	Comments

FIR_R1S => D_FIR1S

DP param	Q003 param	Comments
N	N	Must be GIP.
MX	1 or UNUSED	Must be GIP if used.
NT	NT	Must be GIP.
D	D	Must be GIP.
A	A	
X	X	
Y	Y	

FIR_R1SC => D_FIR1S

DP param	Q003 param	Comments
N	N	Must be GIP.
MX	1 or UNUSED	Must be GIP if used.
NT	NT	Must be GIP.
D	D	Must be GIP.
A	A	
X	X	
Y	Y	

IIR_C1S => D_IIR1S

DP param	Q003 param	Comments
N	N	Must be GIP.
MX	1 or UNUSED	Must be GIP if used.
NZ	NZ	Must be GIP.
NP	NP	Must be GIP.
D	D	Must be GIP.
C	C	
Flag	FG	
X	X	
Y0	Y0	
Y	Y	
Y0′	Y0′	

IIR_C22 => ---

DP param	Q003 param	Comments

MOC_TPSE => D_MTRANS

DP param	Q003 param	Comments
M	N	Must be GIP.
N	M	Must be GIP.
X	X	
Y	Y	

SSP_AGC => D_AGC

DP param	Q003 param	Comments
N	N	Must be GIP.
NI	NI	
FC	FC	
PERIOD	PERIOD	Must be GIP.
PMAX	PMAX	
PMIN	PMIN	
PTARG	PTARG	
FG	FG	
C	C	
X	X	
Y	Y	
FG′	FG′	
C′	C′	
CNT′	CNT′	

SSP_BCOR => ---

DP param	Q003 param	Comments

SSP_BMS => ---

DP param	Q003 param	Comments

SSP_CARD => D_CARD

DP param	Q003 param	Comments
N	N	Must be GIP.
A	A	
B	B	
C	C	
CS	CS	
X	X	
Y	Y	
Z	Z	
CR	CR	

SSP_CCL => ---

DP param	Q003 param	Comments

SSP_CVU => ---

DP param	Q003 param	Comments

SSP_DCD => D_DCD

DP param	Q003 param	Comments
N	N	Must be GIP.
XS	XS	
XC	XC	
X0	X0	
YS	YS	
YC	YC	

SSP_DNS => ---

DP param	Q003 param	Comments

SSP_DSC => ---

DP param	Q003 param	Comments

SSP_EST => ---

DP param	Q003 param	Comments

SSP_FPD => ---

DP param	Q003 param	Comments

SSP_GAG => ---

DP param	Q003 param	Comments

SSP_LPP => ---

DP param	Q003 param	Comments

SSP_LPP2 => ---

DP param	Q003 param	Comments

SSP_LPPA => ---

DP param	Q003 param	Comments

SSP_MAP => ---

DP param	Q003 param	Comments

SSP_MEB => ---

DP param	Q003 param	Comments

SSP_PDF => ---

DP param	Q003 param	Comments

SSP_PPIN => ---

DP param	Q003 param	Comments

SSP_SYNO => D_SYNO

DP param	Q003 param	Comments
N	N	Must be GIP.
FG	FG	
MAGVAR	MAGVAR	
NS	NS	
EW	EW	
OMNI	OMNI	
BEAR	BEAR	

SSP_TDT => ---

DP param	Q003 param	Comments

SSP_TINT => ---

DP param	Q003 param	Comments

SSP_TRK => ---

DP param	Q003 param	Comments

SSP_UTD => ---

DP param	Q003 param	Comments

SSP_ZDT => D_ZDT

DP param	Q003 param	Comments
N	N	Must be GIP.
CF	CF	
BS	BS	
X	X	
F	F	
S	S	
SCF	SCF	
BSY	BSY	
Y	Y	

VCC_VADD => D_VADD

DP param	Q003 param	Comments
N	N	Must be GIP.
X	X	
Y	Y	
Z	Z	

VCC_VDIV => D_VDIV

DP param	Q003 param	Comments
N	N	Must be GIP.
X	X	
Y	Y	
Z	Z	

VCC_VMUL => D_VMUL

DP param	Q003 param	Comments
N	N (1)	Must be GIP.
Flag	1-FG (2)	
X	X (4)	
Y	Y (3)	
Z	Z	

VCM_CTH2 => D_CTH2

The Q003 parameter TN is not used; its effects must be incorporated into parameter T.

DP param	Q003 param	Comments
N	N (1)	Must be GIP.
Flag	FG (2)	
T	T (3)	
TD	TD (4)	
X	X (6)	
Y	Y (7)	
K	K	
Z	Z	
B	B	

VCM_CTHS => D_CTH2

DP param	Q003 param	Comments
N	N	Must be GIP.
Flag	FG	
T	T	
TD	0	
X	X	
Y	Y	
K	K	
Z	Z	
B	B	

VCM_DTH => D_DTH

DP param	Q003 param	Comments
N	N	Must be GIP.
M	M	
PIC	PIC	
W	W	
WIND	WIND	
CTHR1	CTHR1	
CTHR2	CTHR2	
FTHR1	FTHR1	
FTHR2	FTHR2	
R	R	
X	X	
Y	Y	
R′	R′	
Z	Z	
T	T	

VCM_THCC => ---

DP param	Q003 param	Comments

VCM_THCC => ---

DP param	Q003 param	Comments

VCM_THRS => ---

DP param	Q003 param	Comments

VCM_THRST => ---

DP param	Q003 param	Comments

VOC_CONJ => D_CONJ

DP param	Q003 param	Comments
N	N	Must be GIP.
X	X	
Y	Y	

VOC_PWR => D_PWR

DP param	Q003 param	Comments
N	N	Must be GIP.
Ov	UNUSED	
X	X	
Y	Y	
Z	Z	

VOR_ATN2 => D_ATAN2

DP param	Q003 param	Comments
N	N	Must be GIP.
FG	FG	
X	X	
Y	Y	
Z	Z	

VOR_IIND => D_INDX

Either a normal vector or a v_array may be output. Either may be changed to the other by using D_VFILL. For v_array output Q003 parameter KY is not available through an MPIDGen primitive.

DP param	Q003 param	Comments
N	N	Must be GIP.
K	K	
B	B/VB	This may be a v_array if Y is a v_array.
X	X	
Y	Y/UY	

VOR_INDX => D_INDX

DP param	Q003 param	Comments
N	N	Must be GIP.
K	K	
B	B	
X	X	
Y	Y	

VOR_LIN => D_LIN

DP param	Q003 param	Comments
N	N	Must be GIP.
A	A	
B	B	
X	X	
Y	Y	

VOR_LOG => D_LOG

DP param	Q003 param	Comments
N	N	Must be GIP.
BASE	B	
A	A	
B	C	
X	X	
Y	Y	

VOR_MAG => D_MAG

DP param	Q003 param	Comments
N	N	Must be GIP.
X	X	
Y	Y	

VOR_VCC2 => D_VCC2

DP param	Q003 param	Comments
N	N	Must be GIP.
UR	REPLU	
LR	REPLL	
UL	CLIPU	
LL	CLIPL	
X	X	
Y	Y	

VOR_VIND => D_INDX

DP param	Q003 param	Comments
N	N	Must be GIP.
K	UNUSED	
B	B	
X	X	
Y	Y	

VOR_VSQR => D_SQRT

DP param	Q003 param	Comments
N	N	Must be GIP.
X	X	
Y	Y	

VOR_ZCC => D_ZCC

DP param	Q003 param	Comments
N	N	Must be GIP.
B	UNUSED	
X	X	
Y	Y	
Z	UNUSED	

VRC_MUL => D_VMUL

DP param	Q003 param	Comments
N	N	Must be GIP.
Flag	1-FG	Must be GIP.
X	X	
Y	Y	
Z	Z	

VRR_INP => D_VINP

DP param	Q003 param	Comments
N	N	Must be GIP.
X	X	
Y	Y	
Z	Z	

VRR_VADD => D_VADD

DP param	Q003 param	Comments
N	N	Must be GIP.
X	X	
Y	Y	
Z	Z	

VRR_VDIV => D_VDIV

DP param	Q003 param	Comments
N	N	Must be GIP.
X	X	
Y	Y	
Z	Z	

VRR_VMUL => D_VMUL

DP param	Q003 param	Comments
N	N	Must be GIP.
Flag	UNUSED	
X	X	
Y	Y	
Z	Z	

VRR_VSUB => D_VSUB

DP param	Q003 param	Comments
N	N	Must be GIP.
Flag	UNUSED	
X	X	
Y	Y	
Z	Z	

Appendix D - Partition Graphs - Iconic Form

Portable Reusable Application Software
SBIR Phase I Final Report

October 28, 1998

Prepared by:
Management Communciations and Control, Inc. (MCCI)
2000 North Fourteenth Street, Suite 220
Arlington, VA 22201
Under Contract N68335-98-C-0140

Appendix D. Partition Graphs - Iconic Form

This Appendix contains the iconic form of each Partition Graph for the DICASS sonobuoy processing application ported from the AN/UYS-2 implementation to the MCCI Autocoding Toolset implementation.

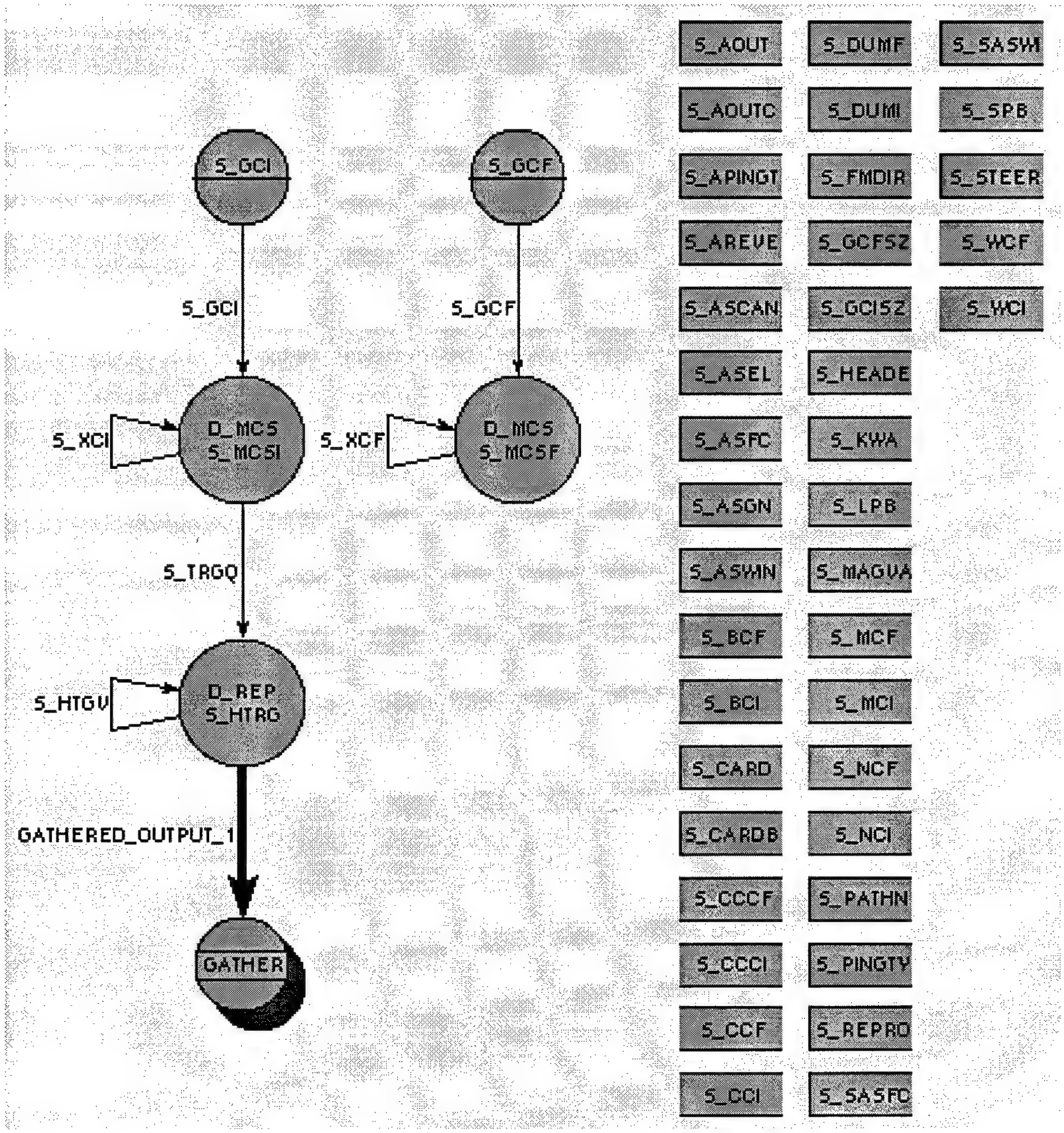


Figure 1. Partition P_INF

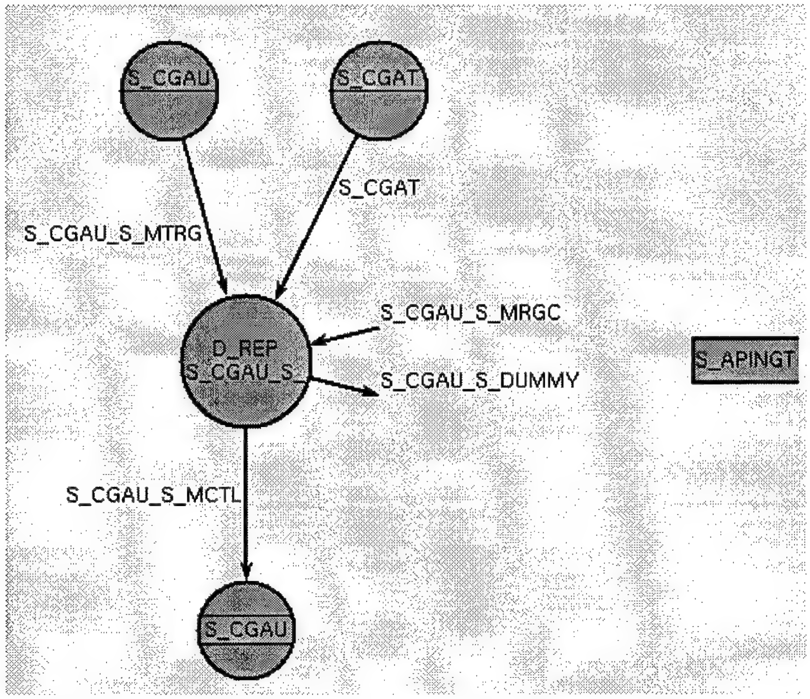


Figure 2. Partition P_INF_1

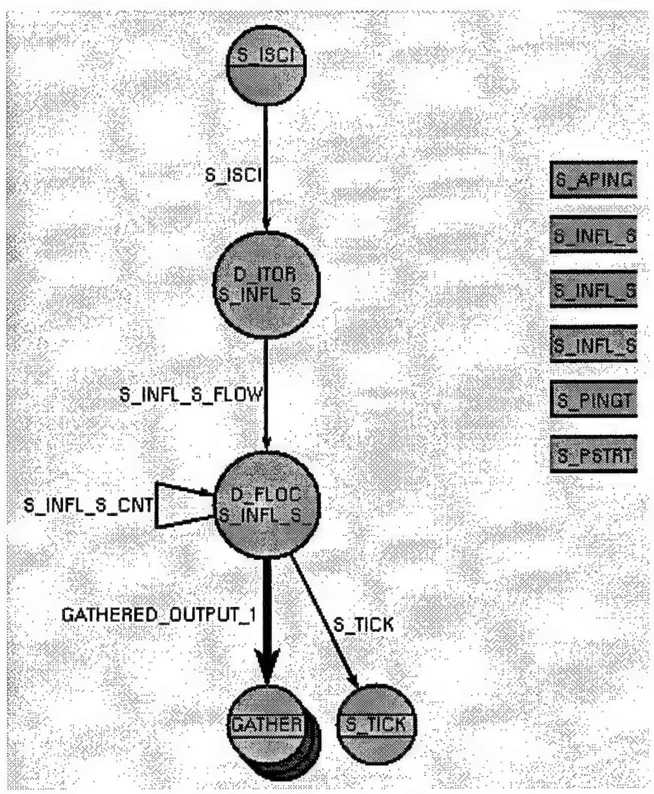


Figure 3. Partition P_INF_2

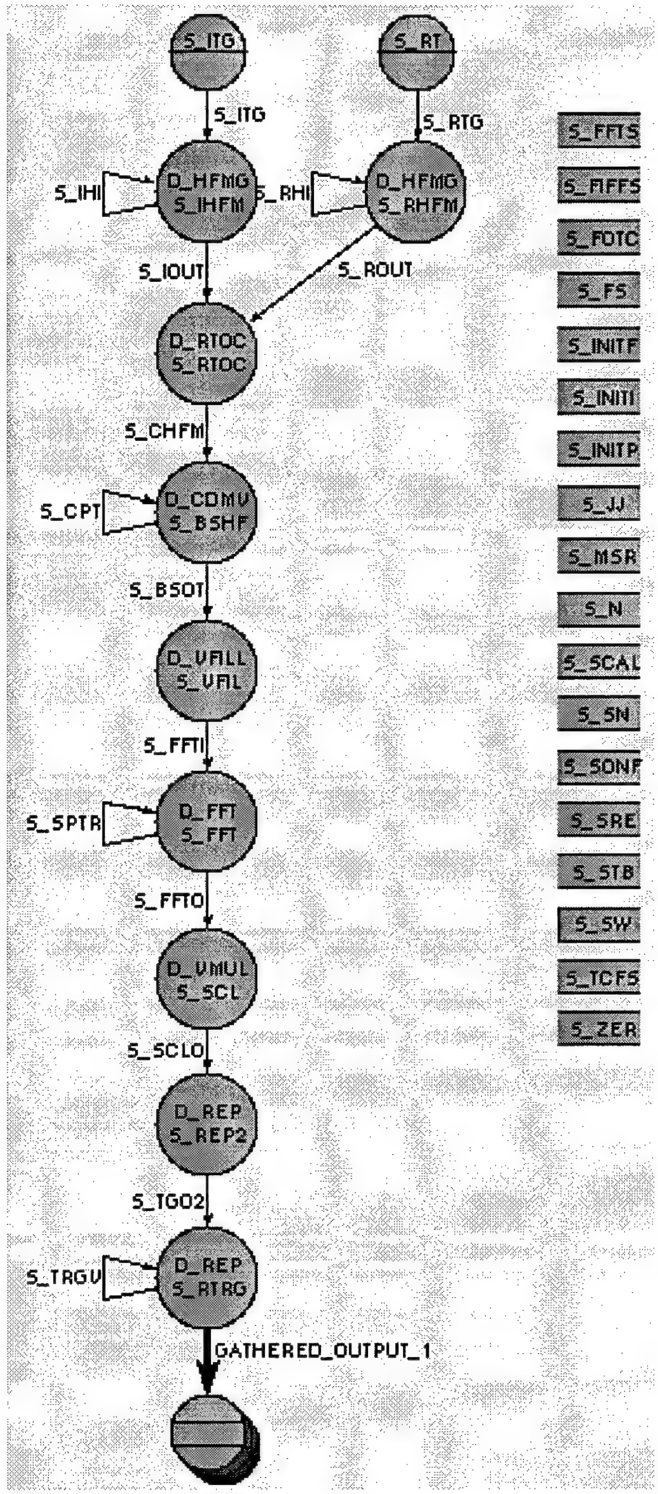


Figure 4. Partition P_INF_3

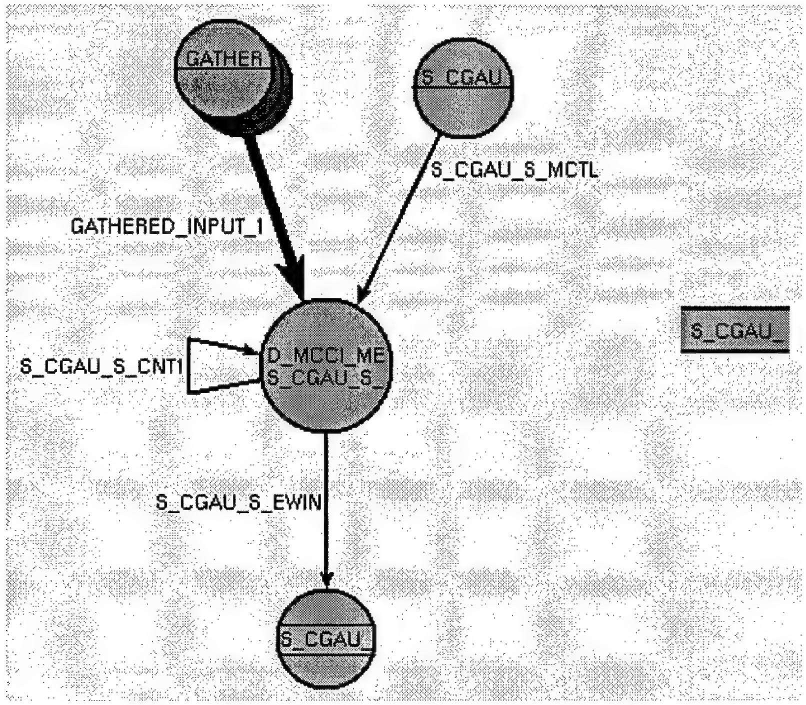


Figure 5. Partition P_CGA_1

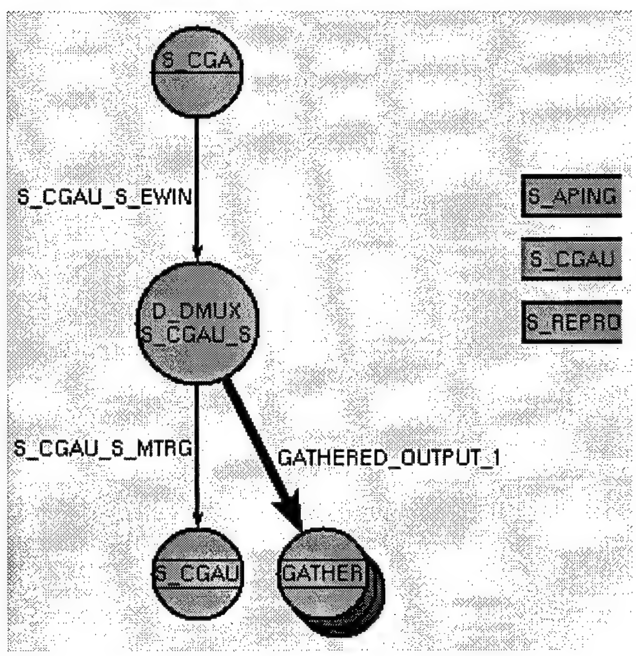


Figure 6. Partition CGA_2

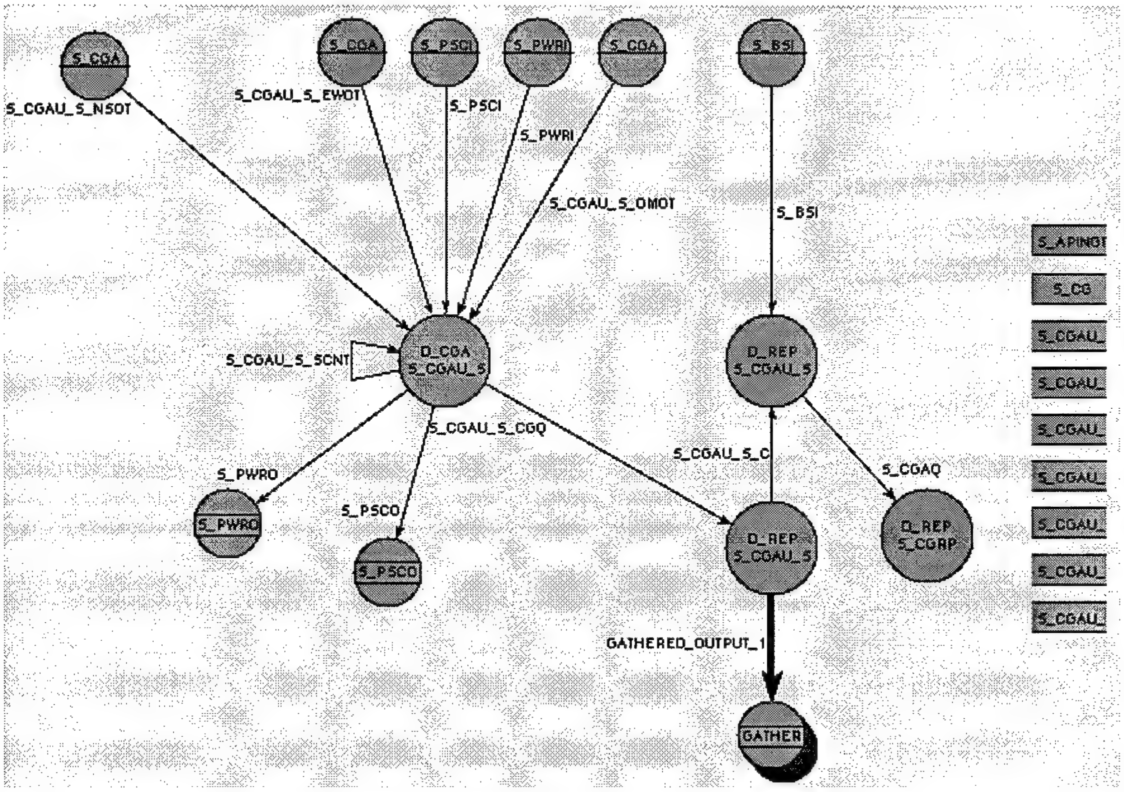


Figure 7. Partition P_CGA_3

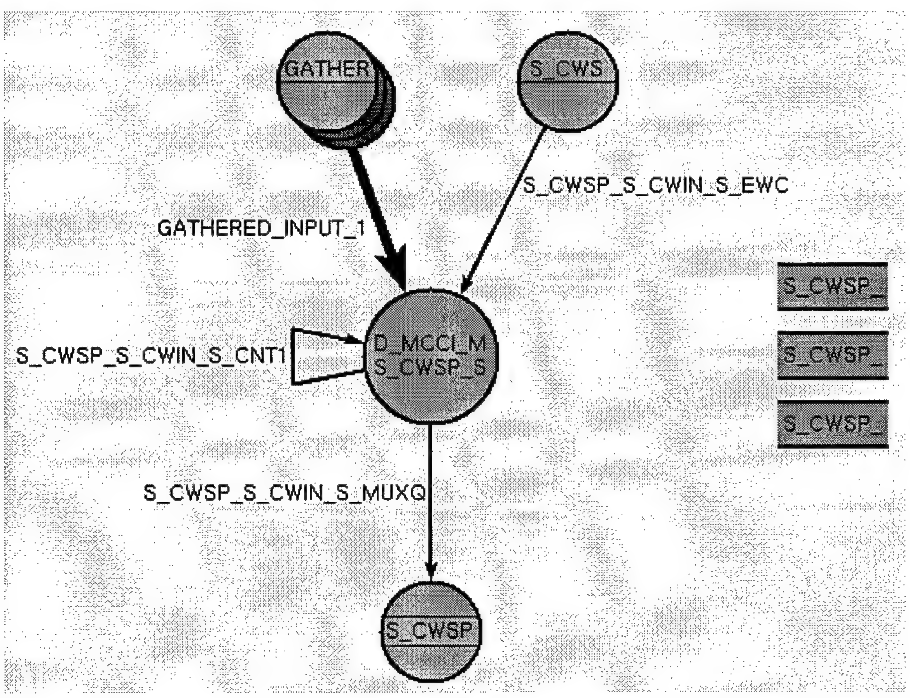


Figure 8. Partition P_CWSIN_1

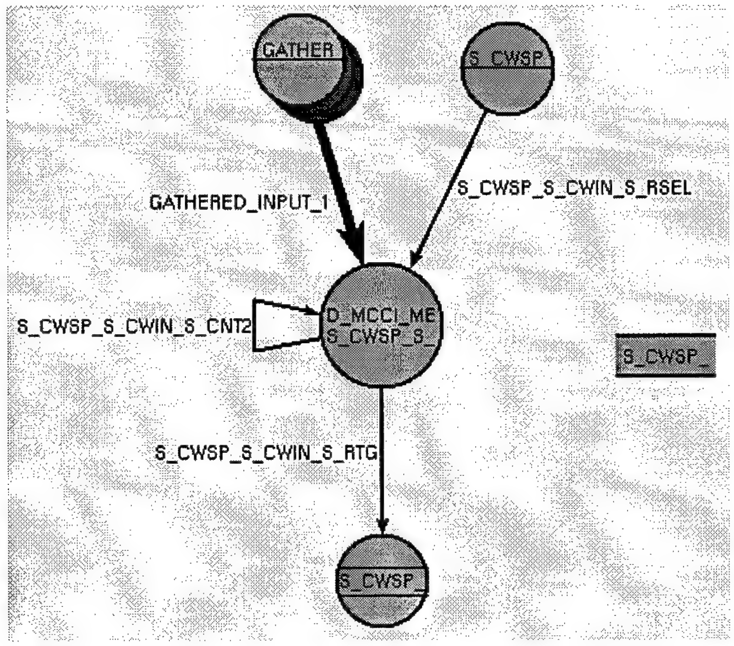


Figure 9. Partition P_CWSIN_2

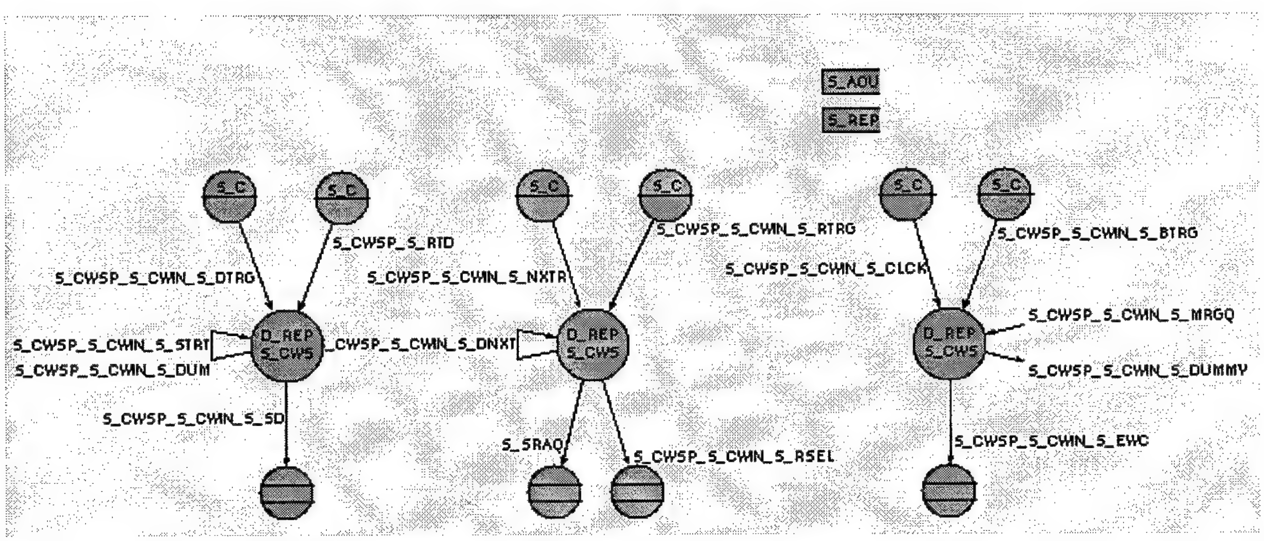


Figure 10. Partition P_CWSIN_3

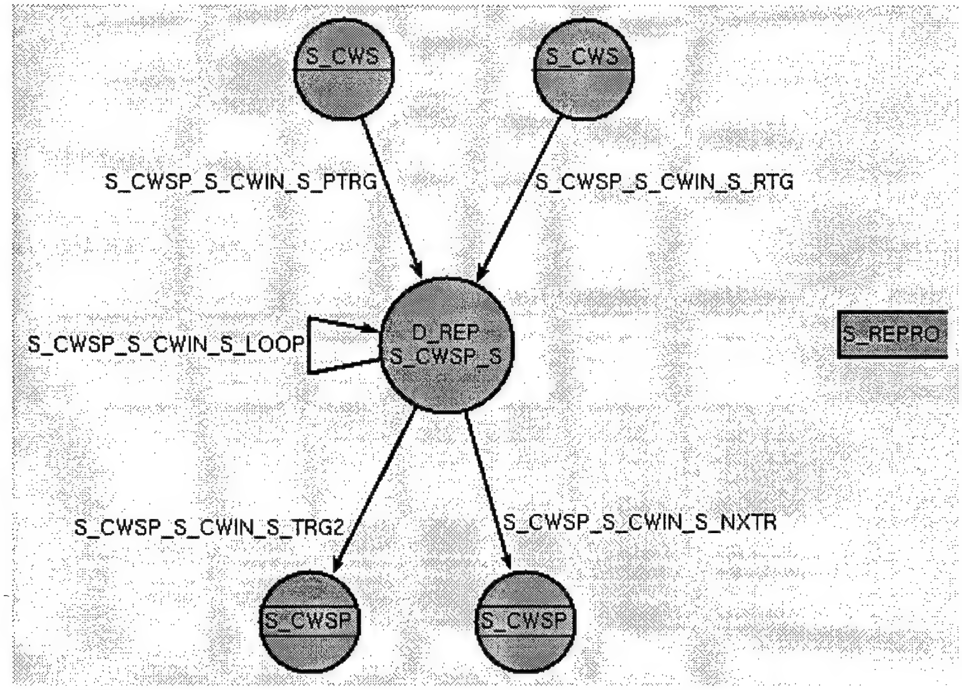


Figure 11. Partition P_CWSIN_3B

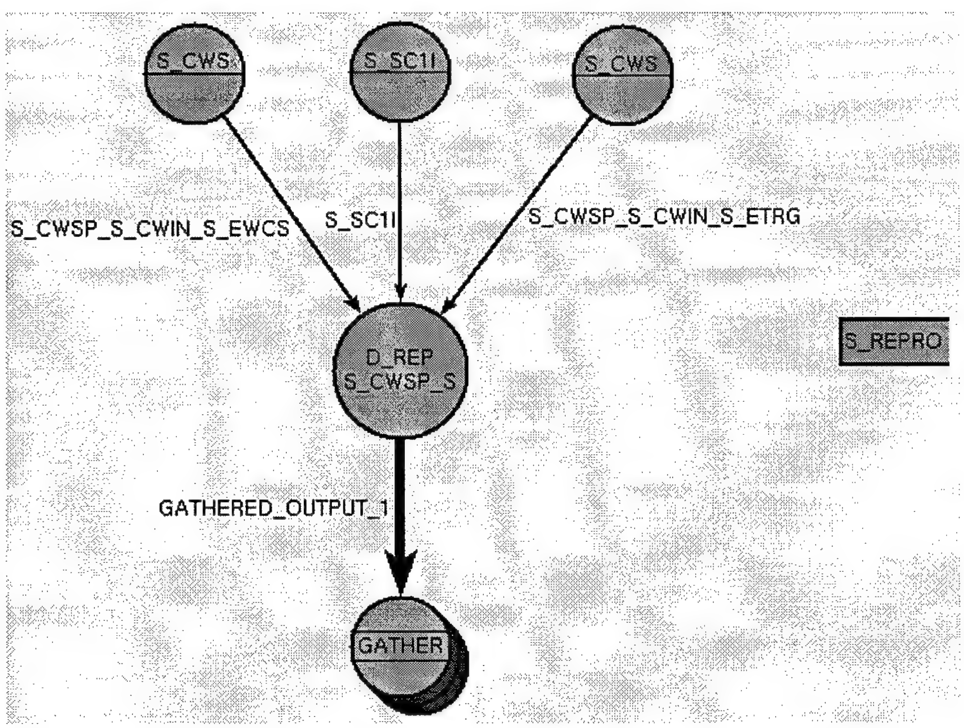


Figure 12. Partition P_CWSIN_3C

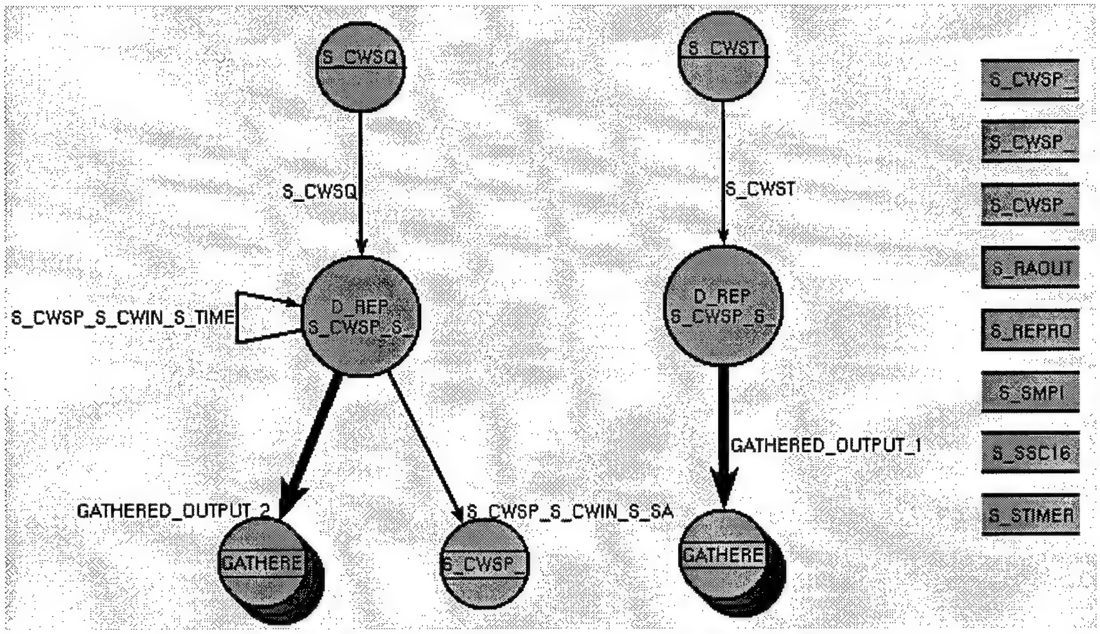


Figure 13. Partition P_CWSIN_3D

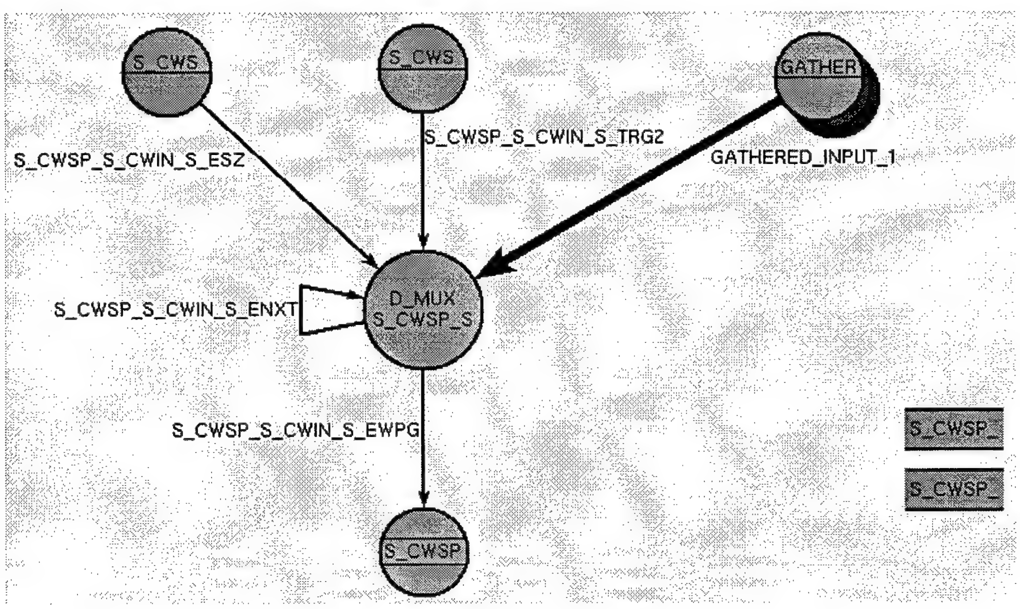


Figure 14. Partition P_CWSIN_3E

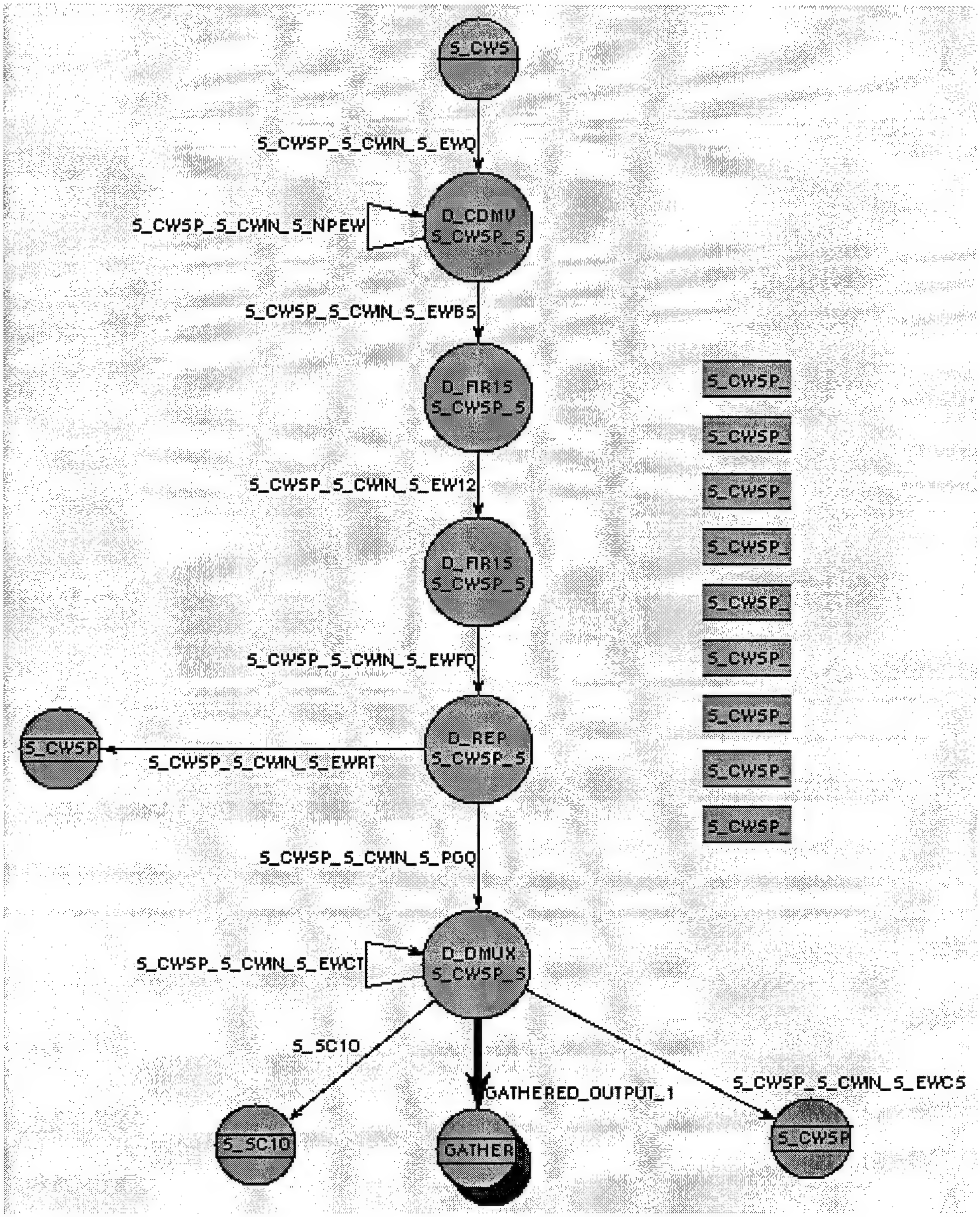


Figure 15. Partition P_CWSIN_4

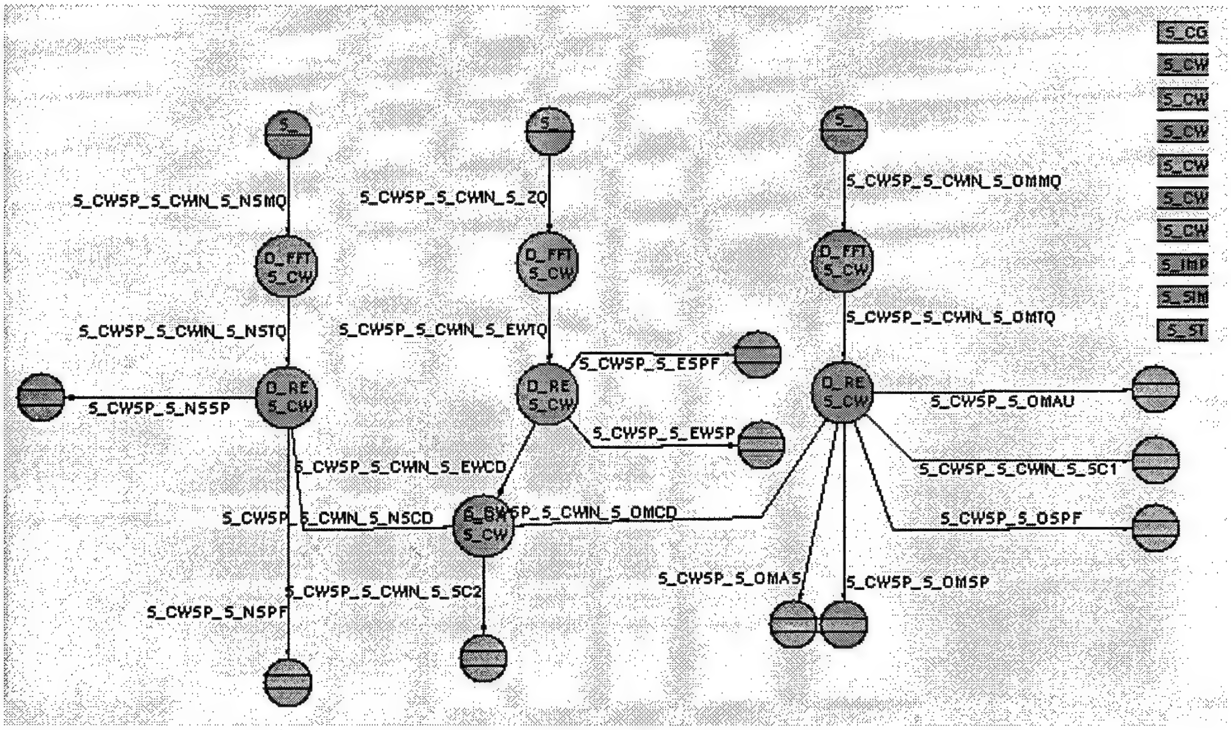


Figure 16. Partition P_CWSIN_5

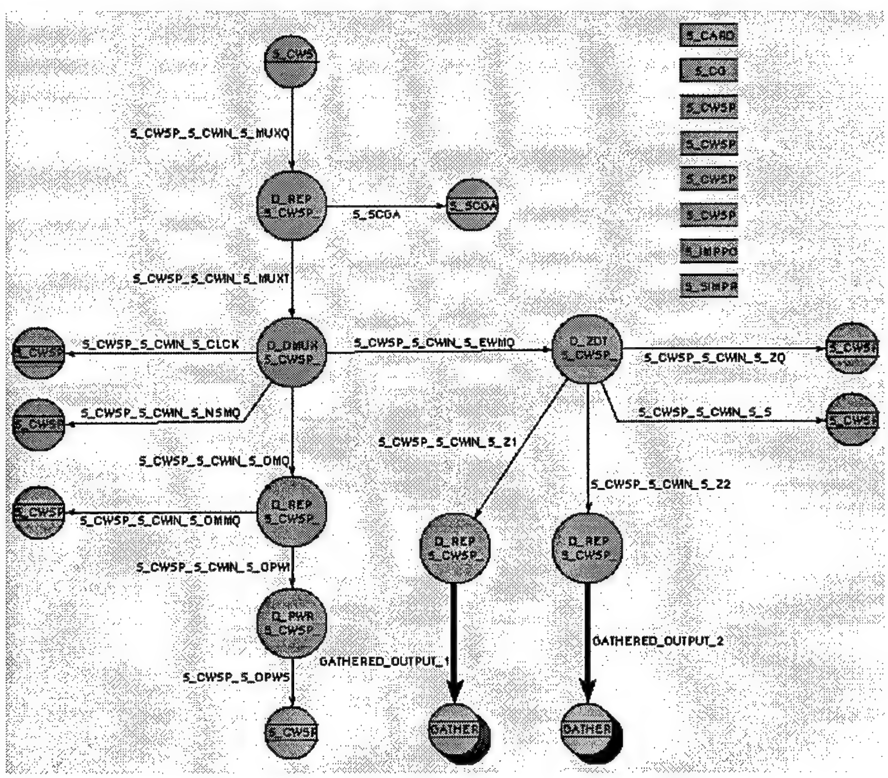


Figure 17. Partition P_CWSIN_5B

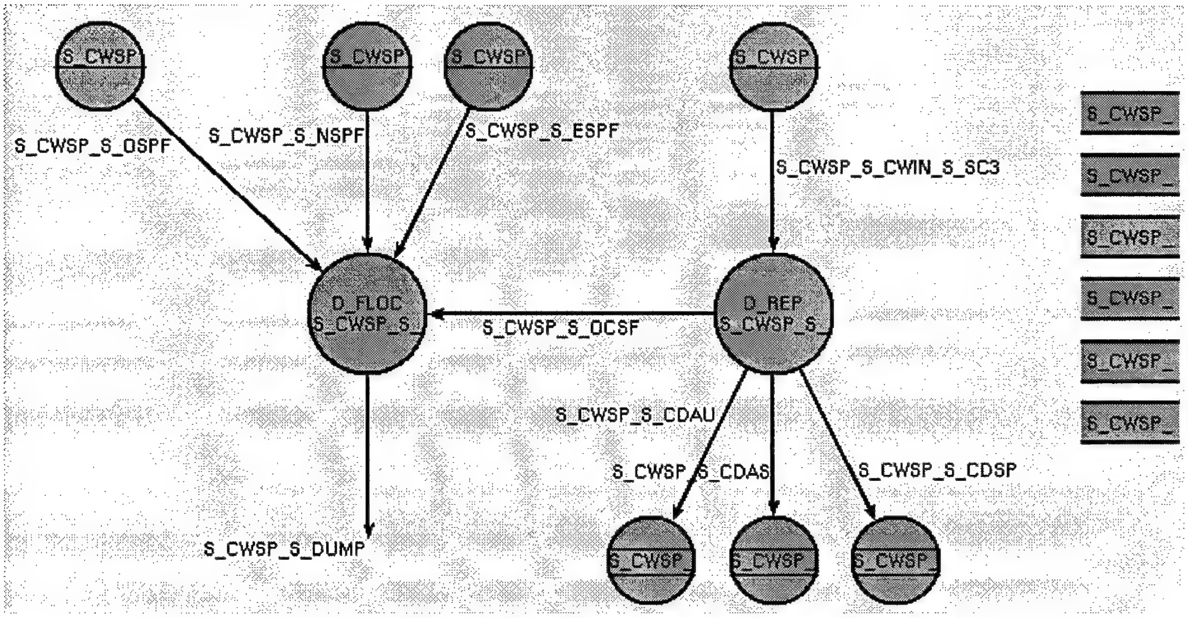


Figure 18. Partition P_CWSIN_6

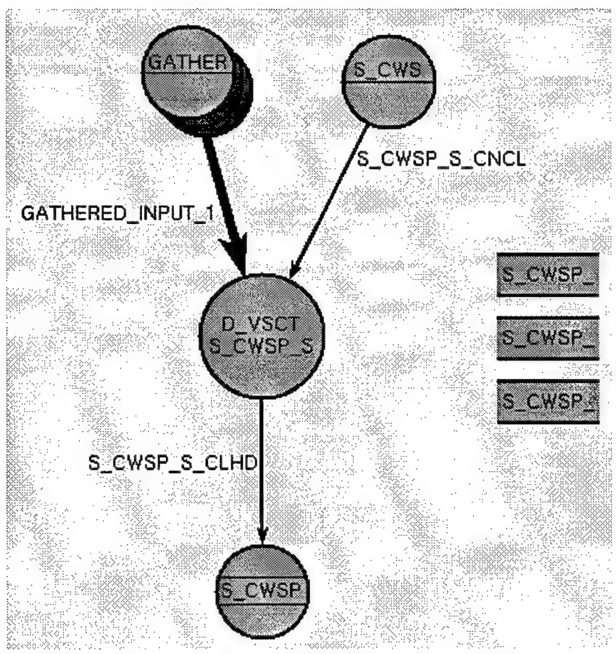


Figure 19. Partition P_CWSIN_7

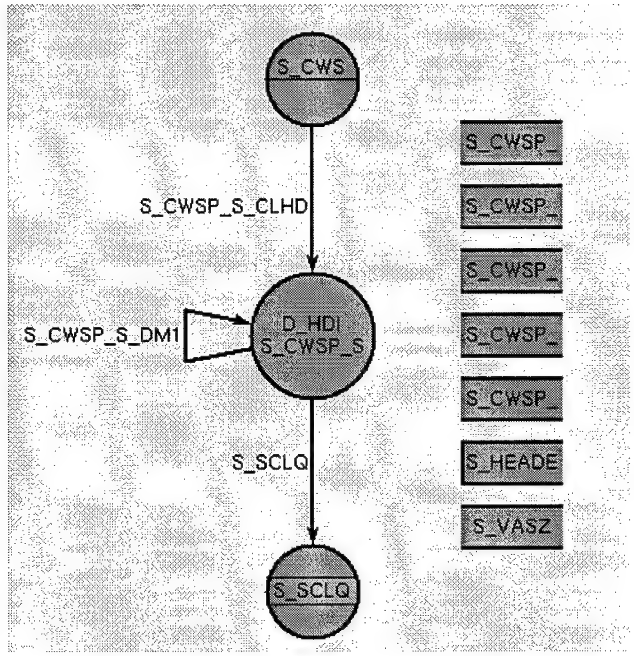


Figure 20. Partition P_CWSIN_7A

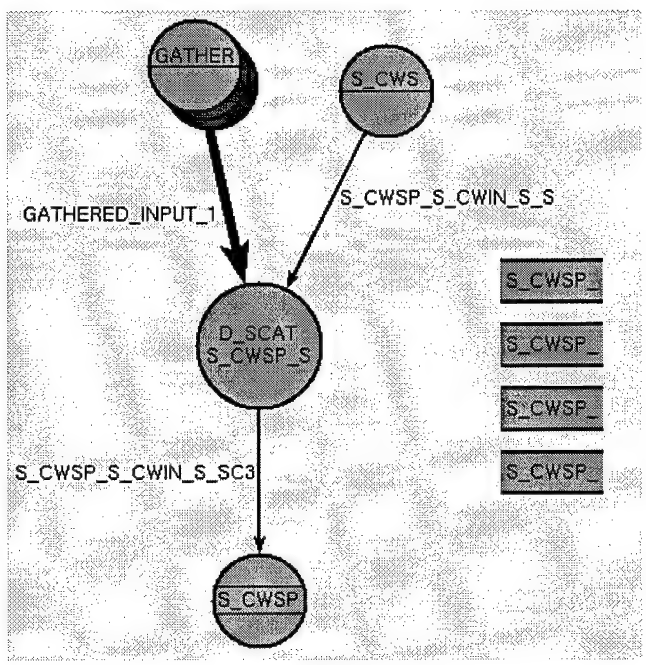


Figure 21. Partition P_CWSIN_8

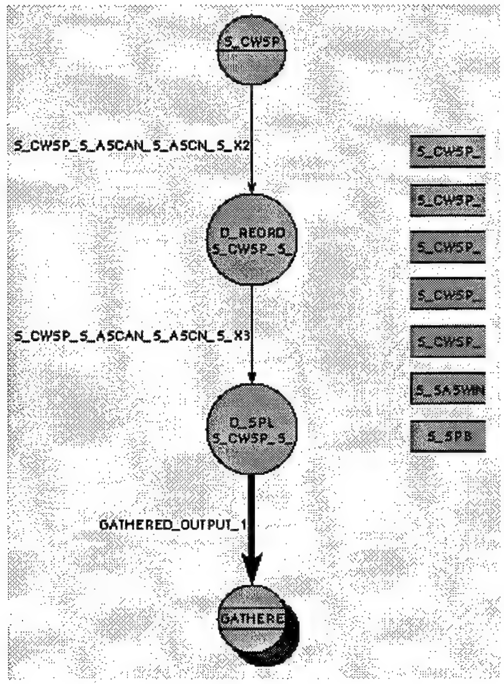


Figure 22. Partition P_CWS_1

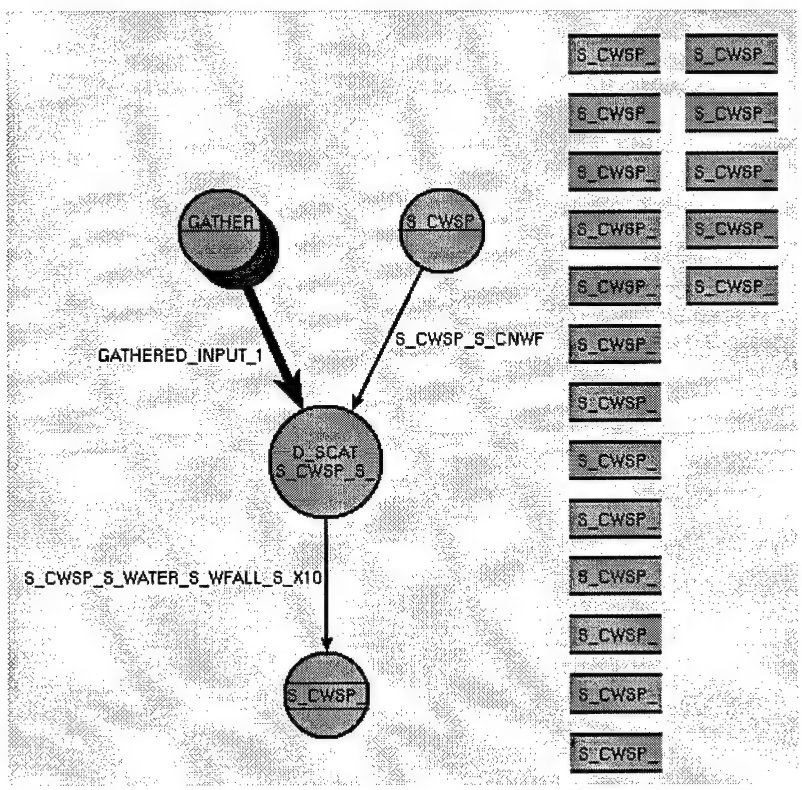


Figure 23. Partition P_CWS_1A

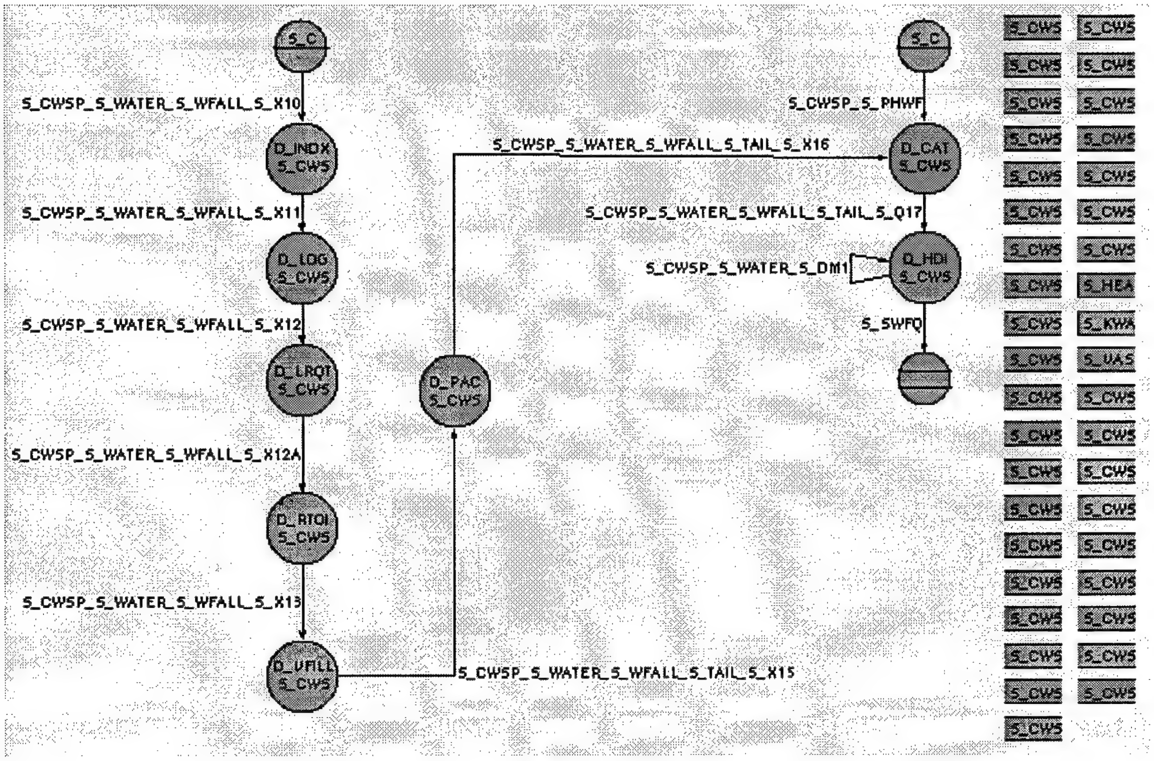


Figure 24. Partition P_CWS_1B

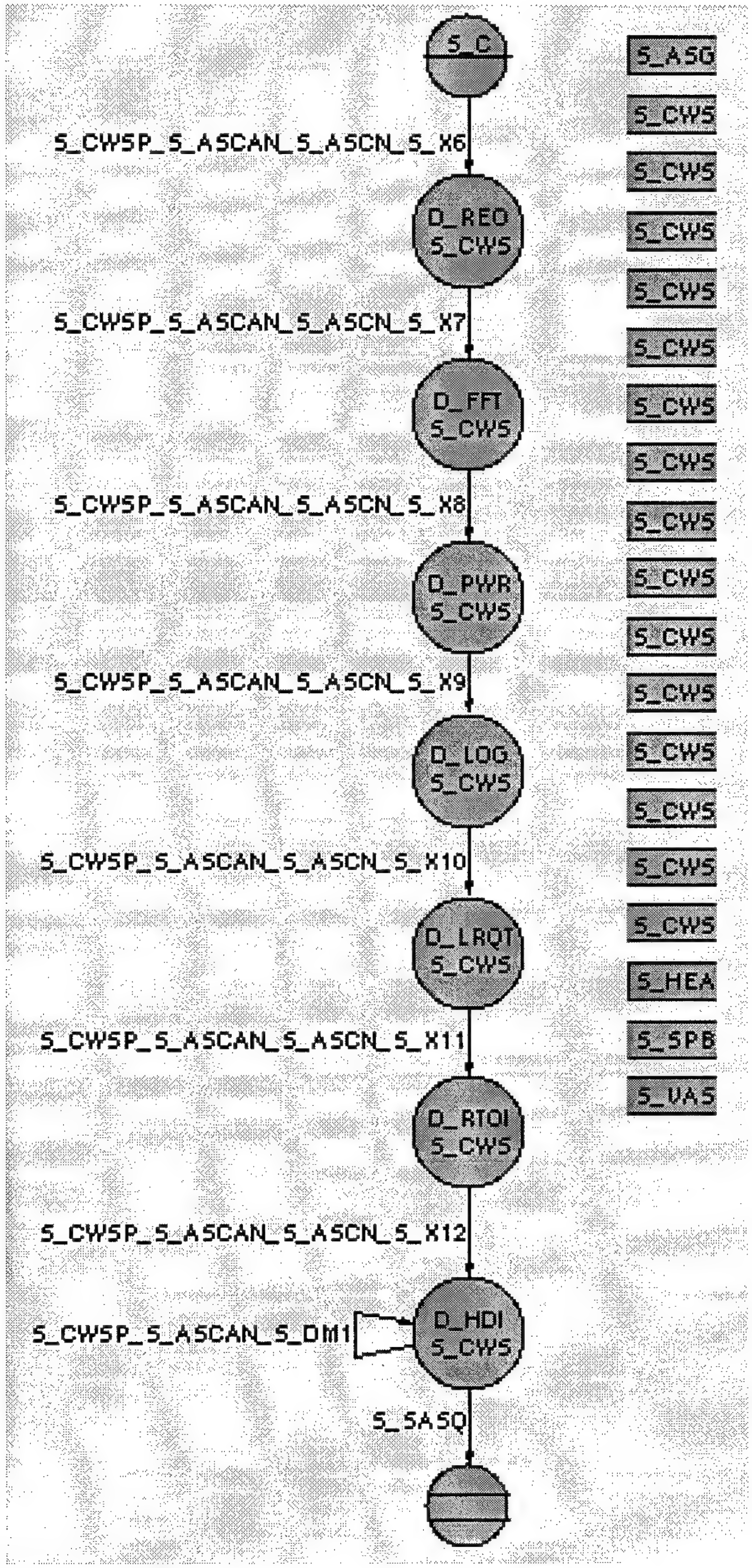


Figure 25. Partition P_CWS_1E

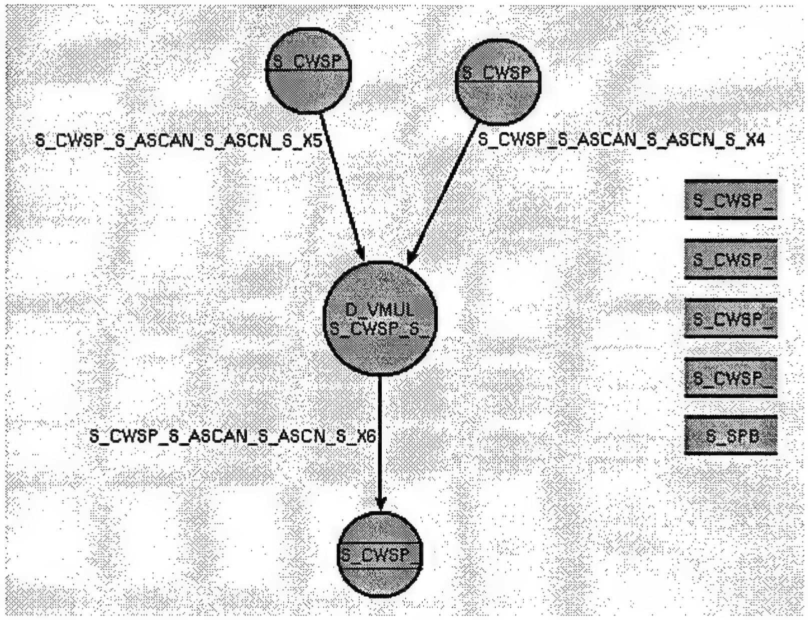


Figure 26. Partition P_CWS_1F

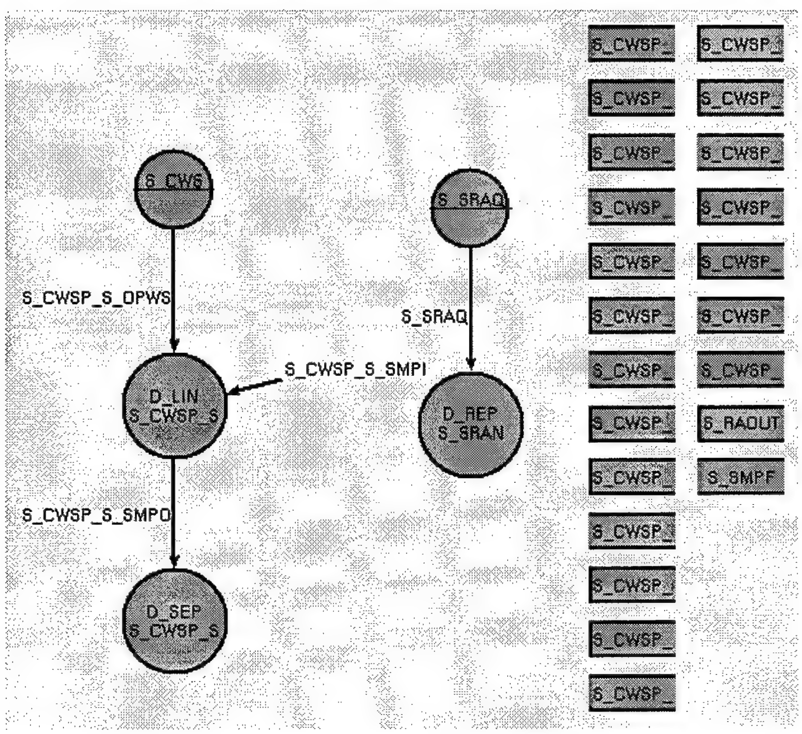


Figure 27. Partition P_CWS_1K

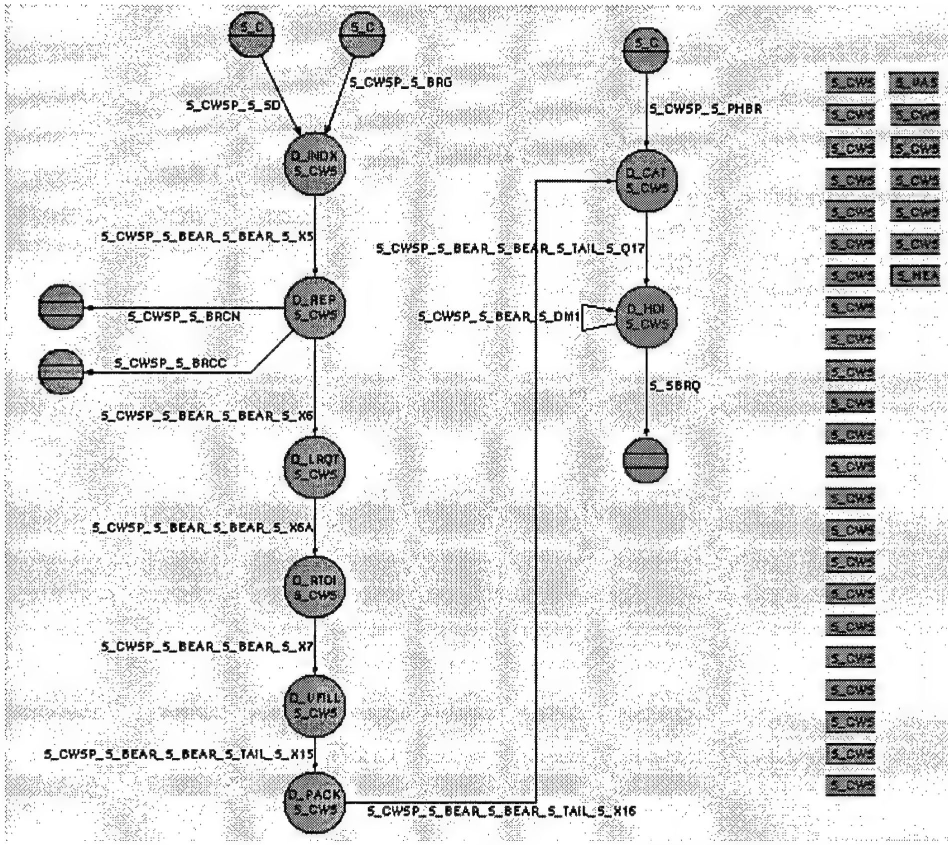


Figure 28. Partition P_CWS_1L

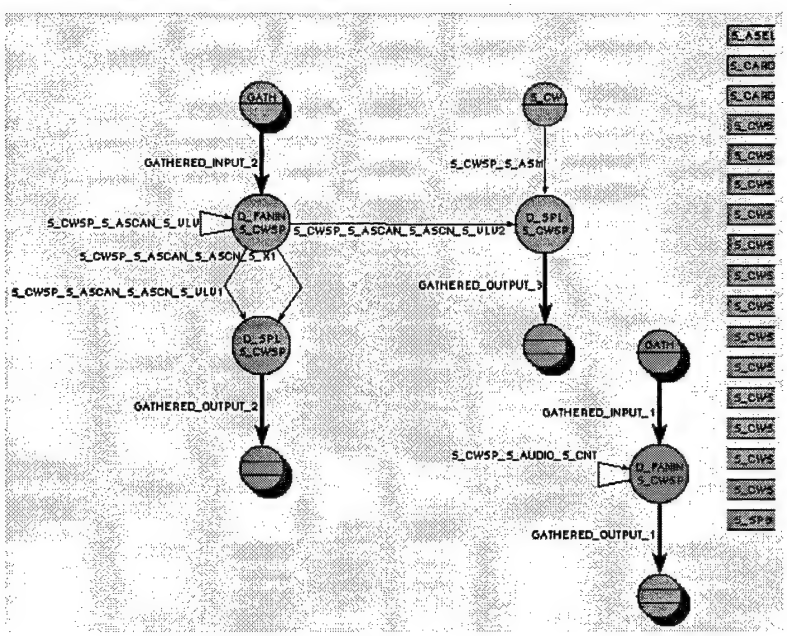


Figure 29. Partition P_CWS_2

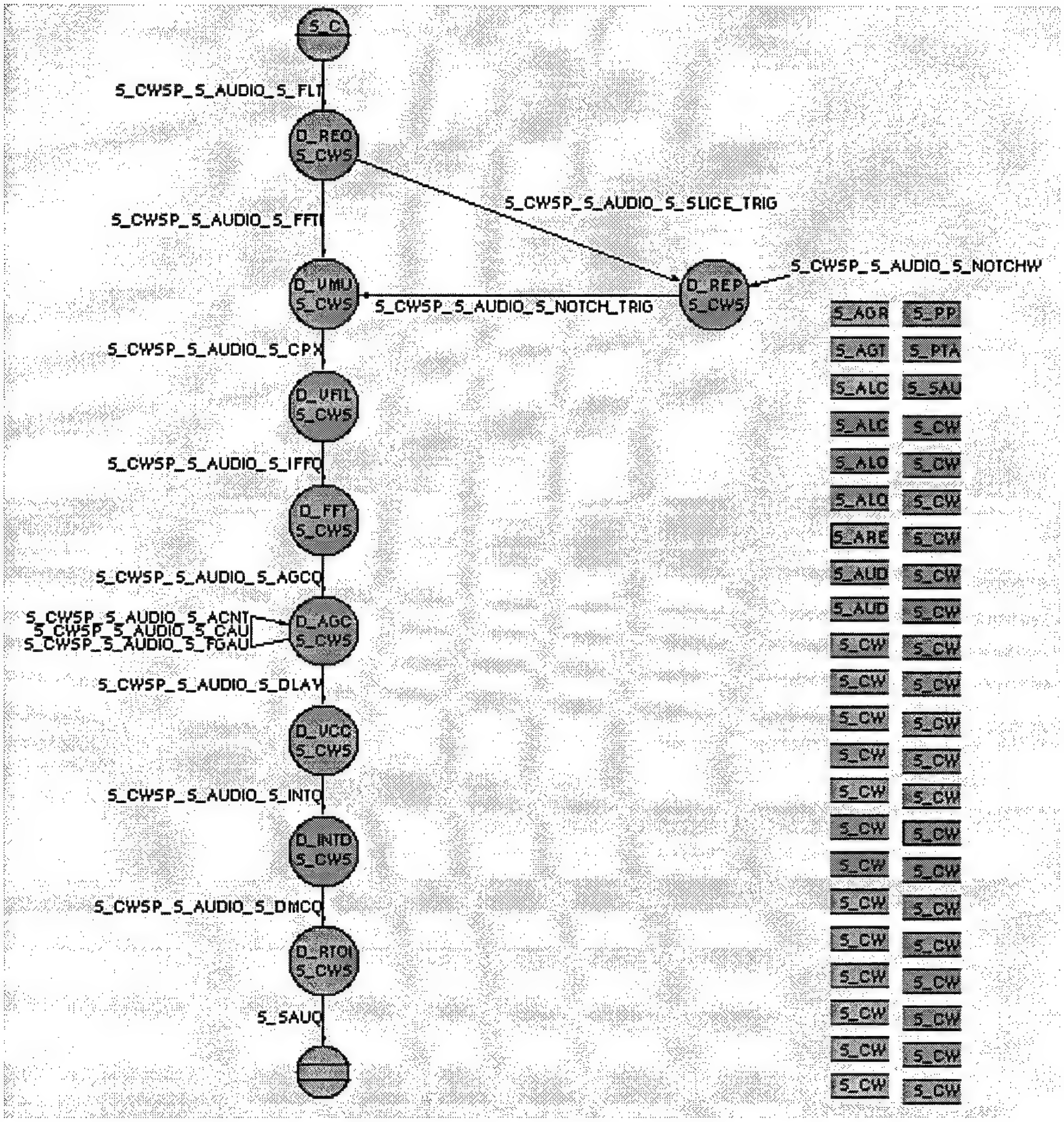


Figure 30. Partition P_CWS_2B

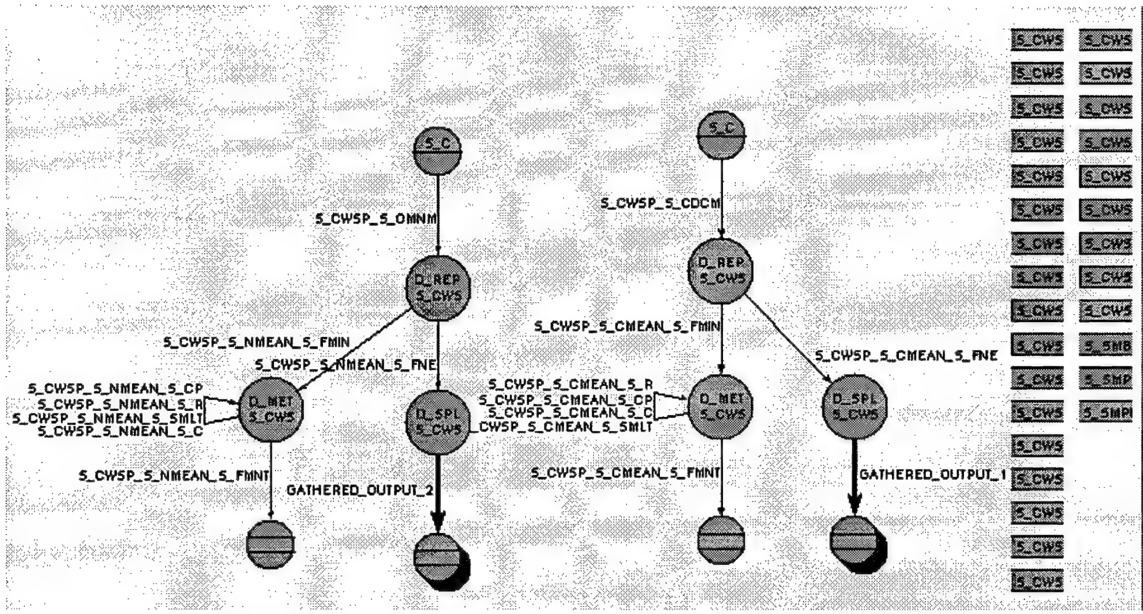


Figure 31. Partition P_CWS_3

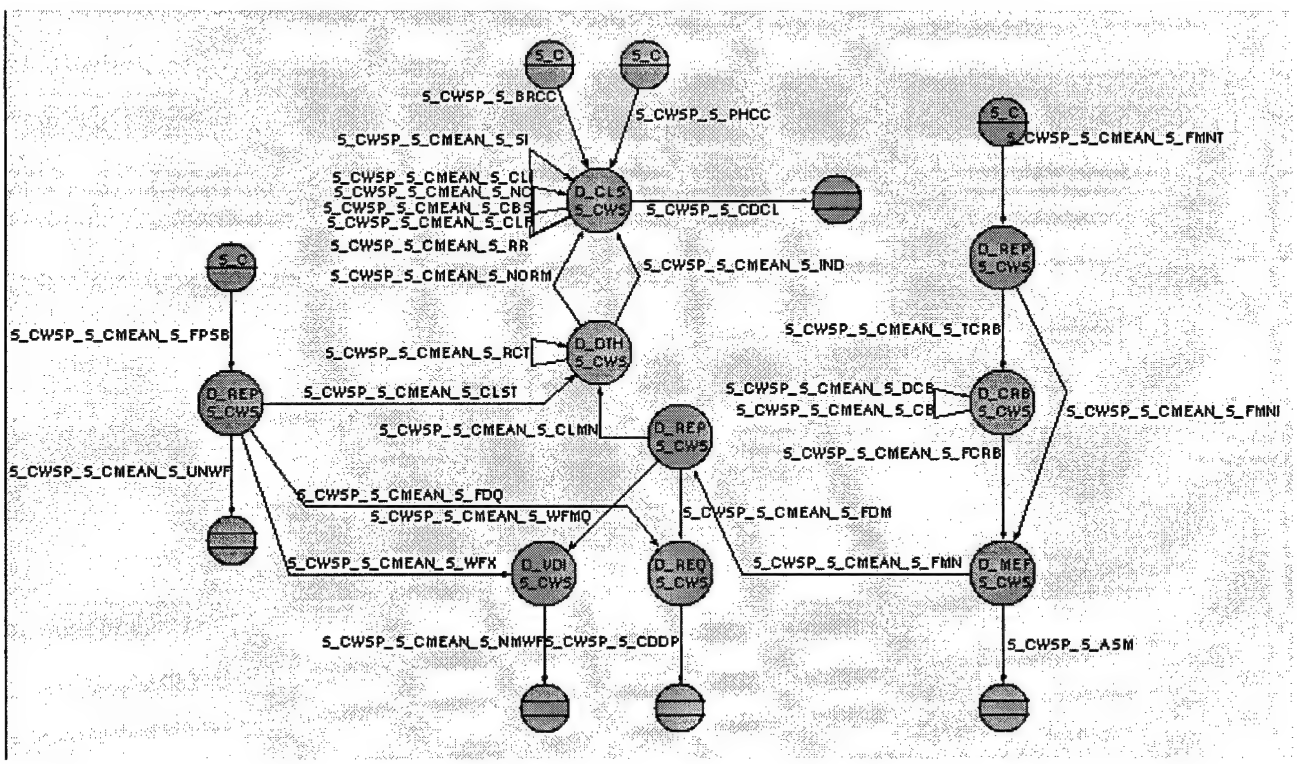


Figure 32. Partition P_CWS_3B

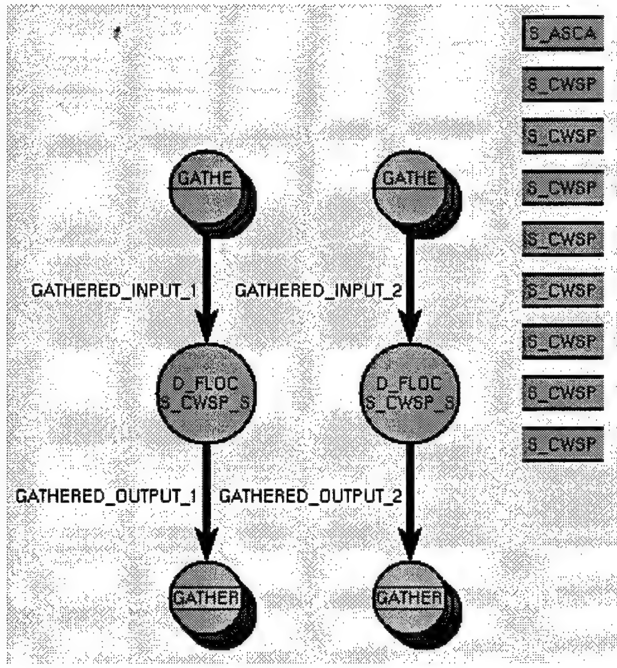


Figure 33. Partition P_CWS_3C

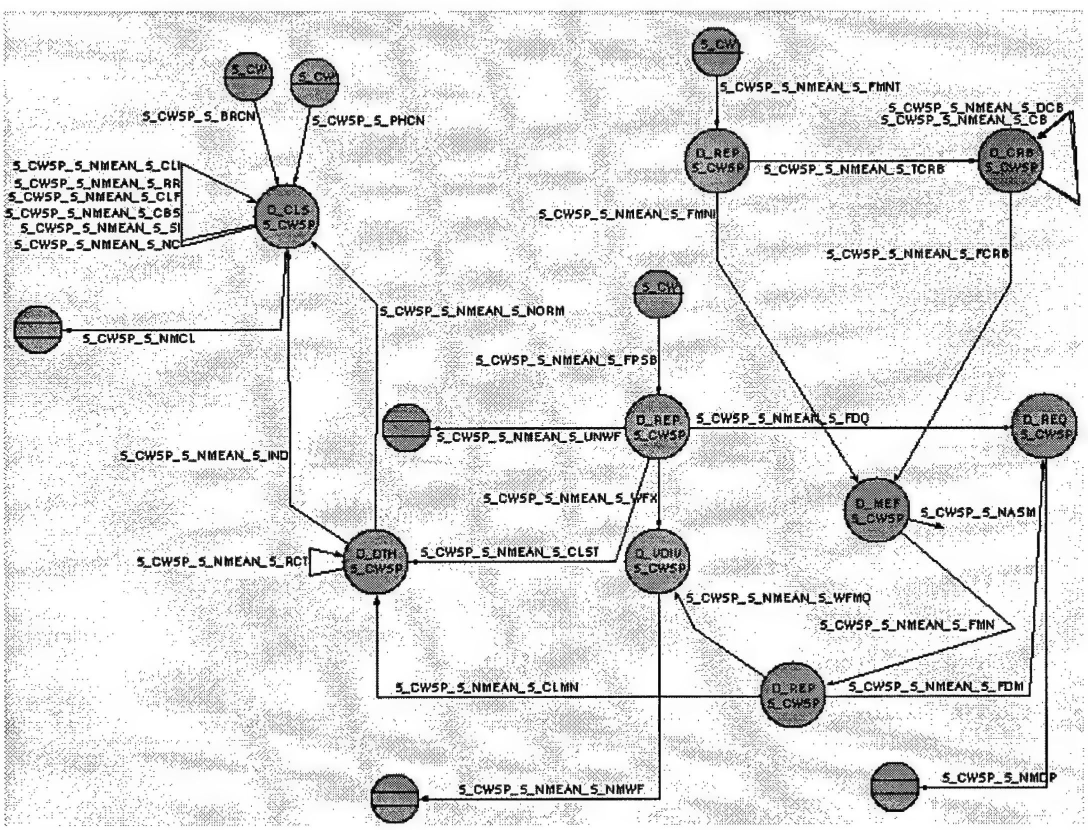


Figure 34. Partition P_CWS_3D

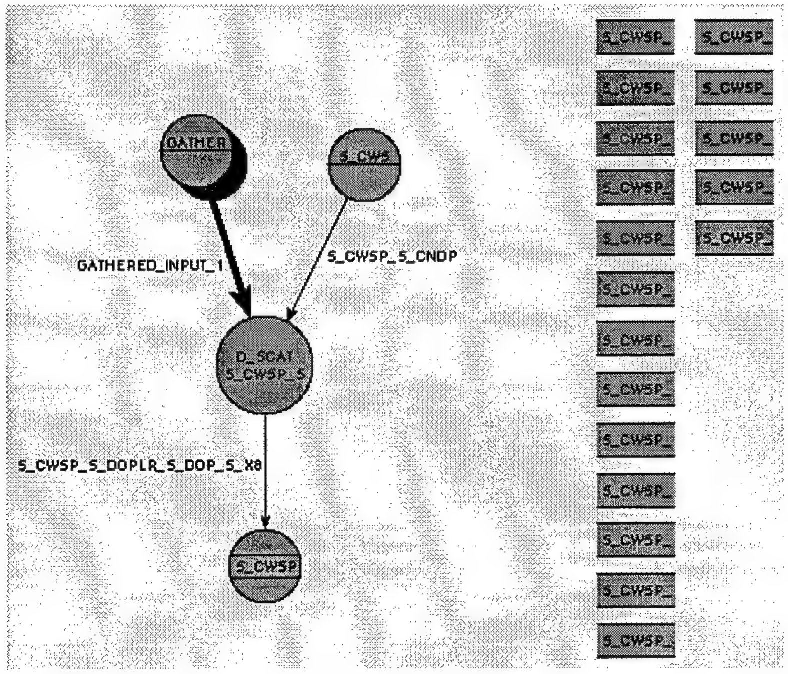


Figure 35. Partition P_CWS_3E

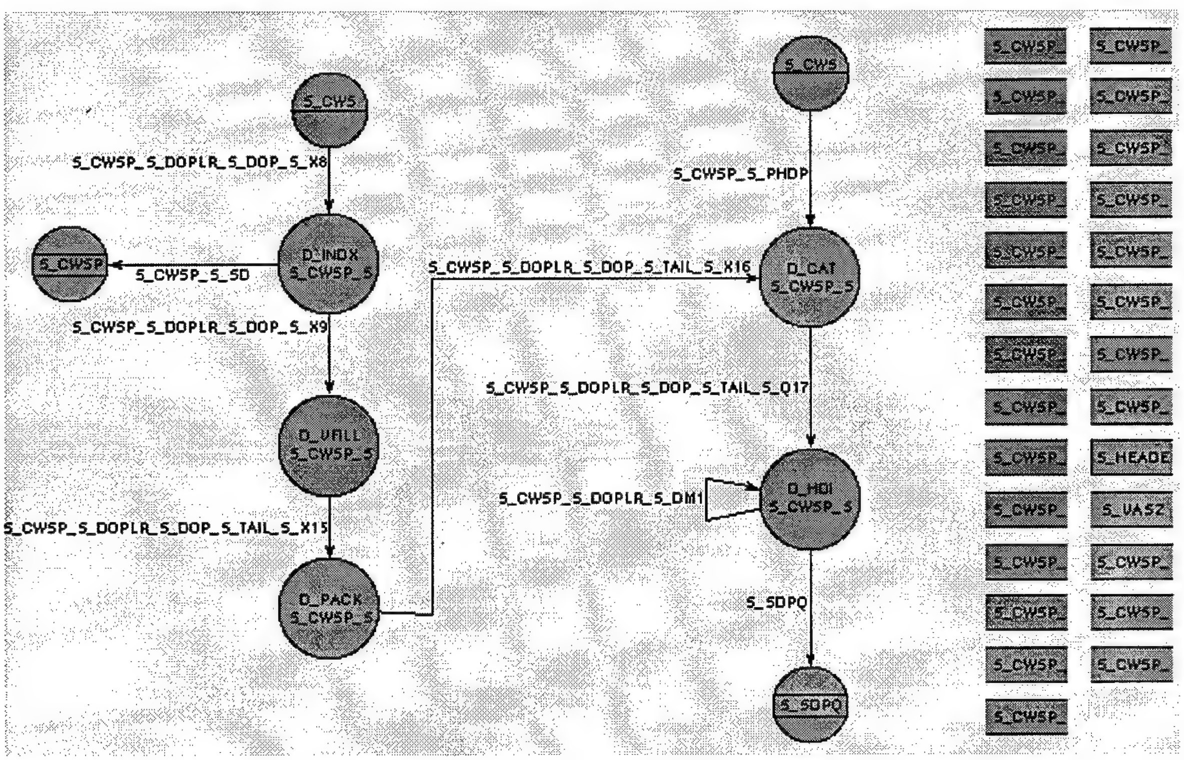


Figure 36. Partition P_CWS_3F

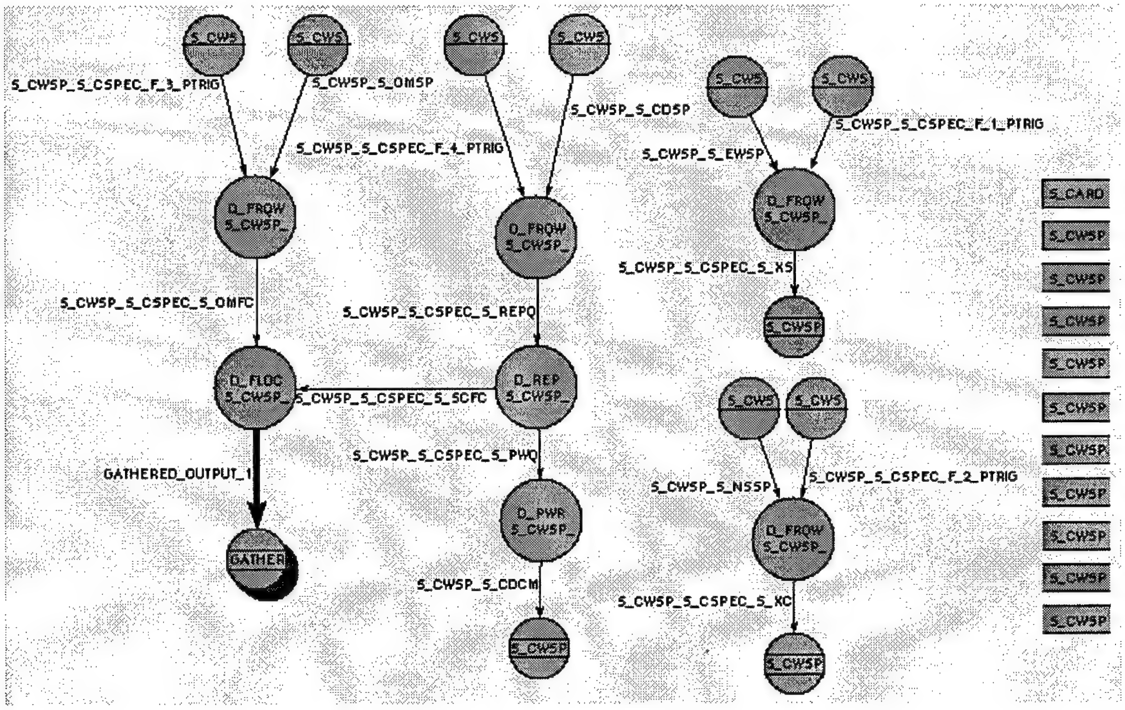


Figure 37. Partition P_CWS_4

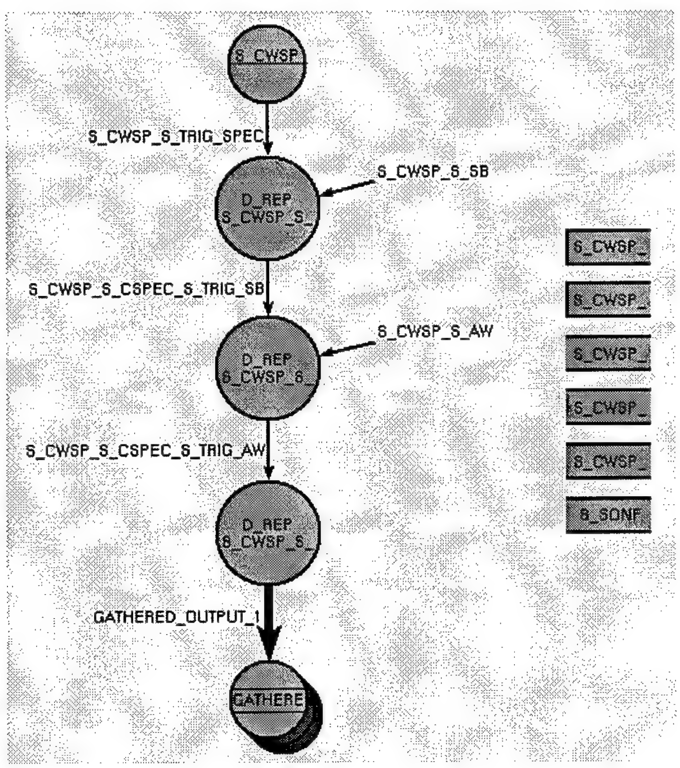


Figure 38. Partition P_CWS_4A

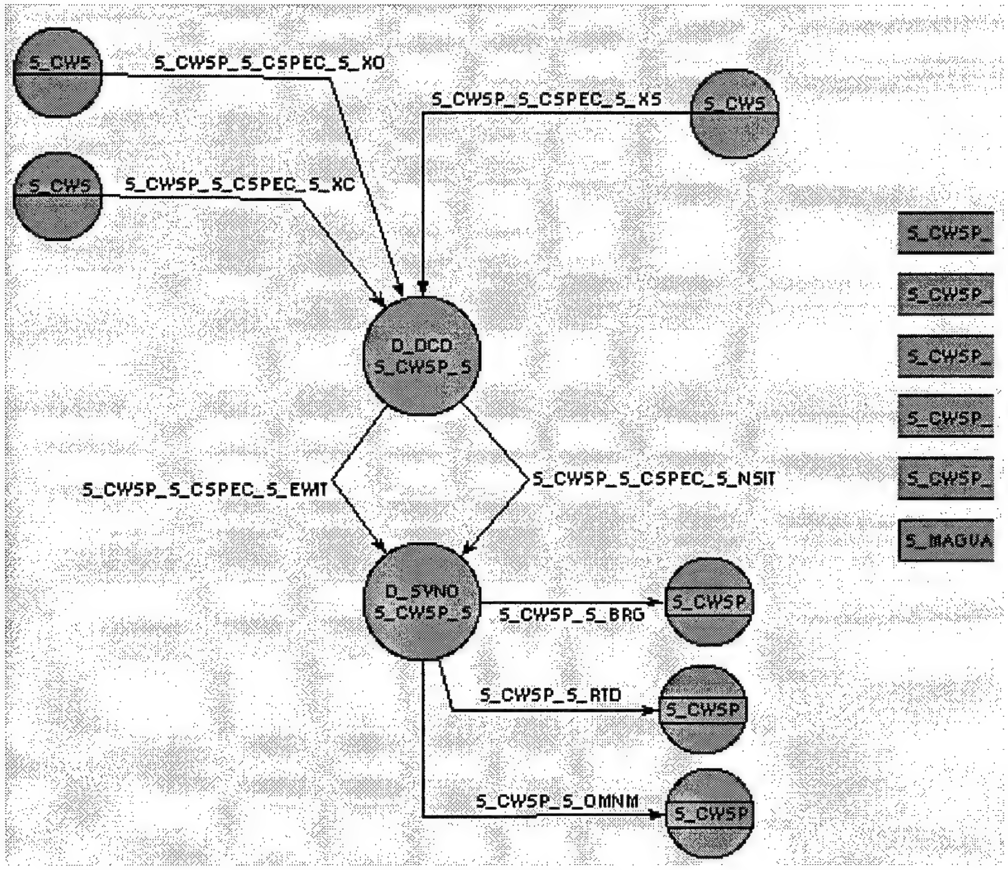


Figure 39. Partition P_CWS_4B

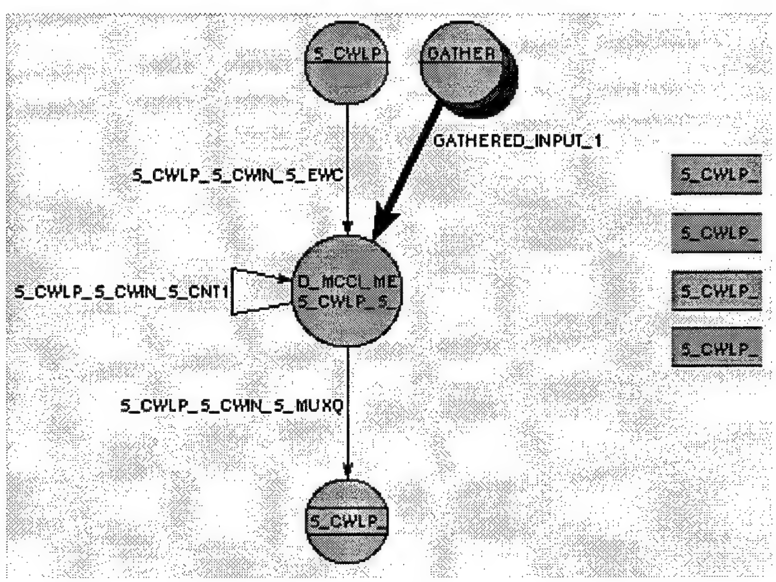


Figure 40. Partition P_CWLIN_1

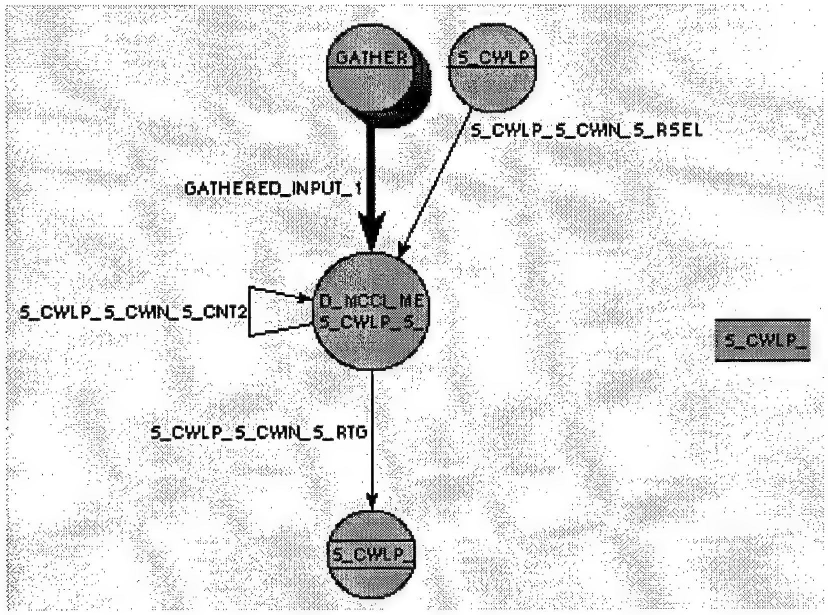


Figure 41. Partition P_CWLIN_2

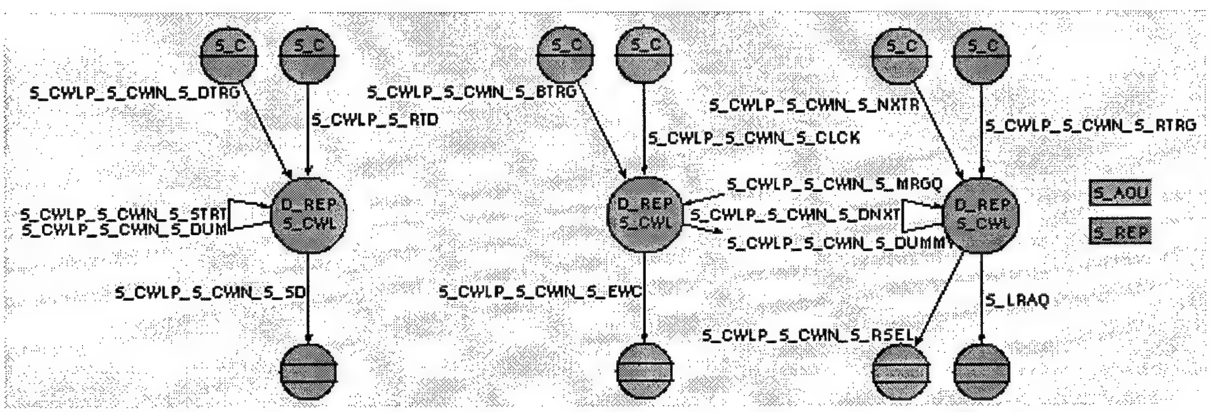


Figure 42. Partition P_CWLIN_3

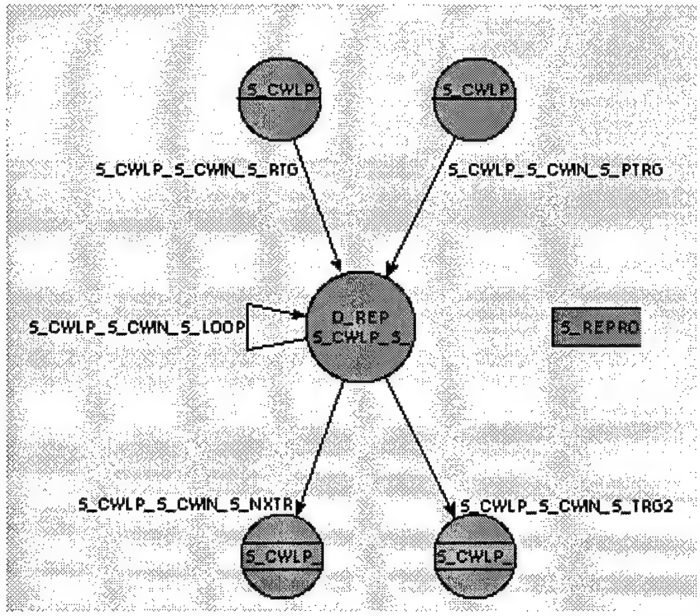


Figure 43. Partition P_CWLIN_3B

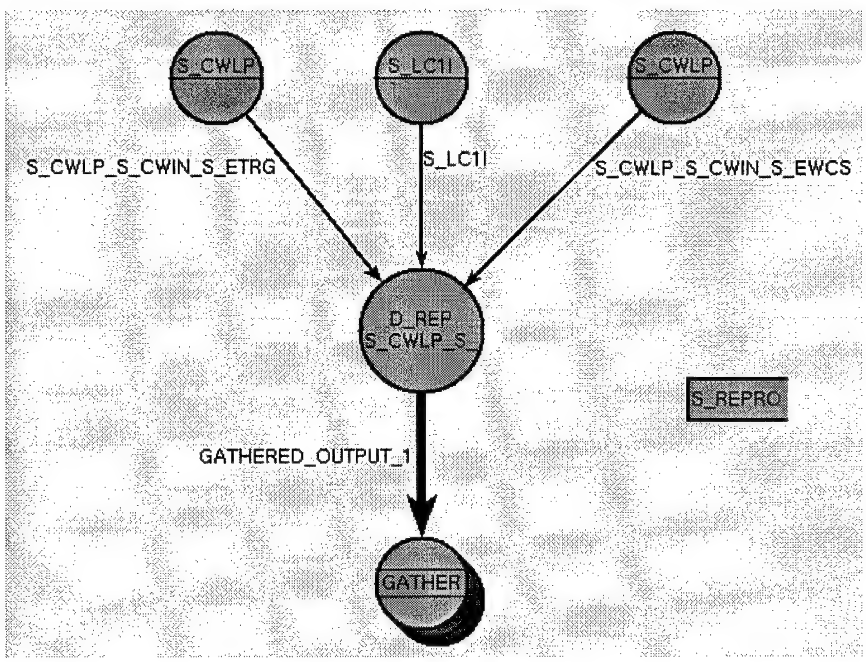


Figure 44. Partition P_CWLIN_3C

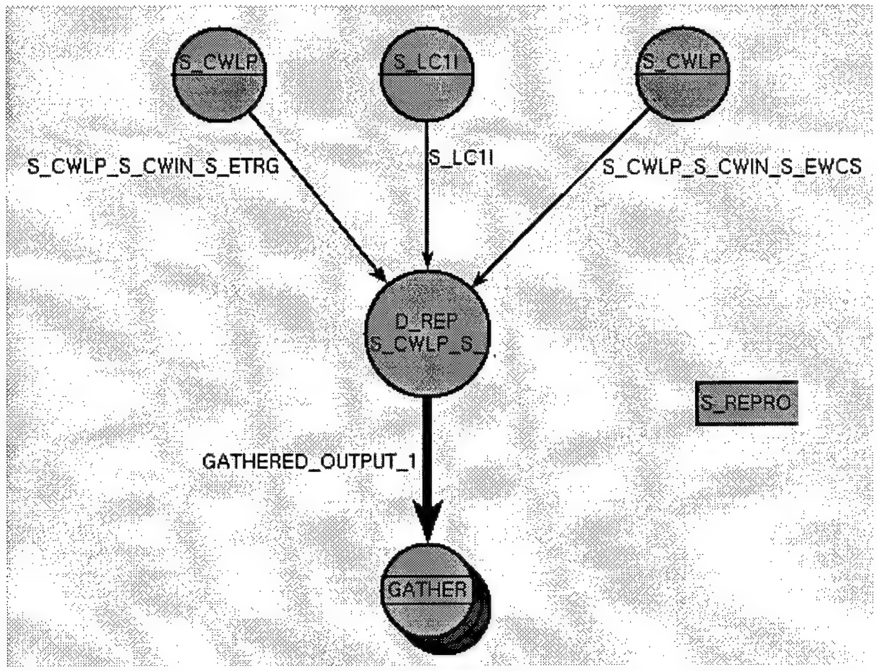


Figure 45. Partition P_CWLIN_3D

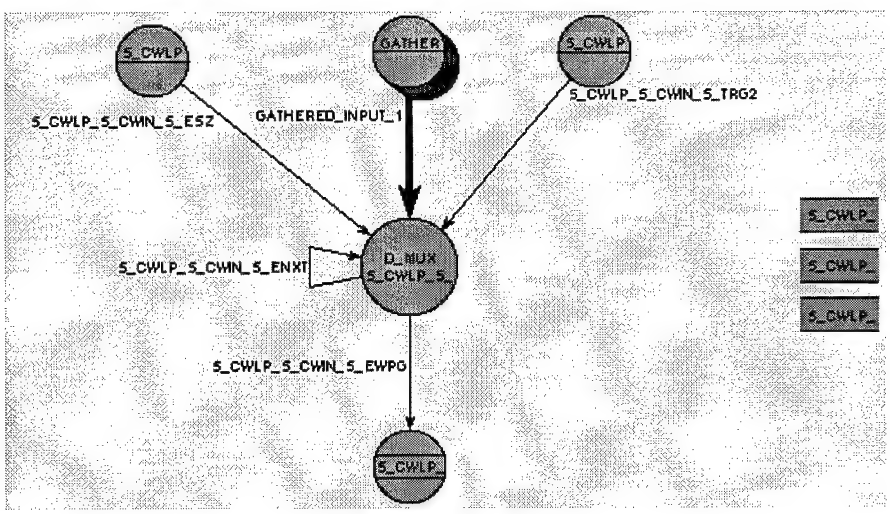


Figure 46. Partition P_CWLIN_3E

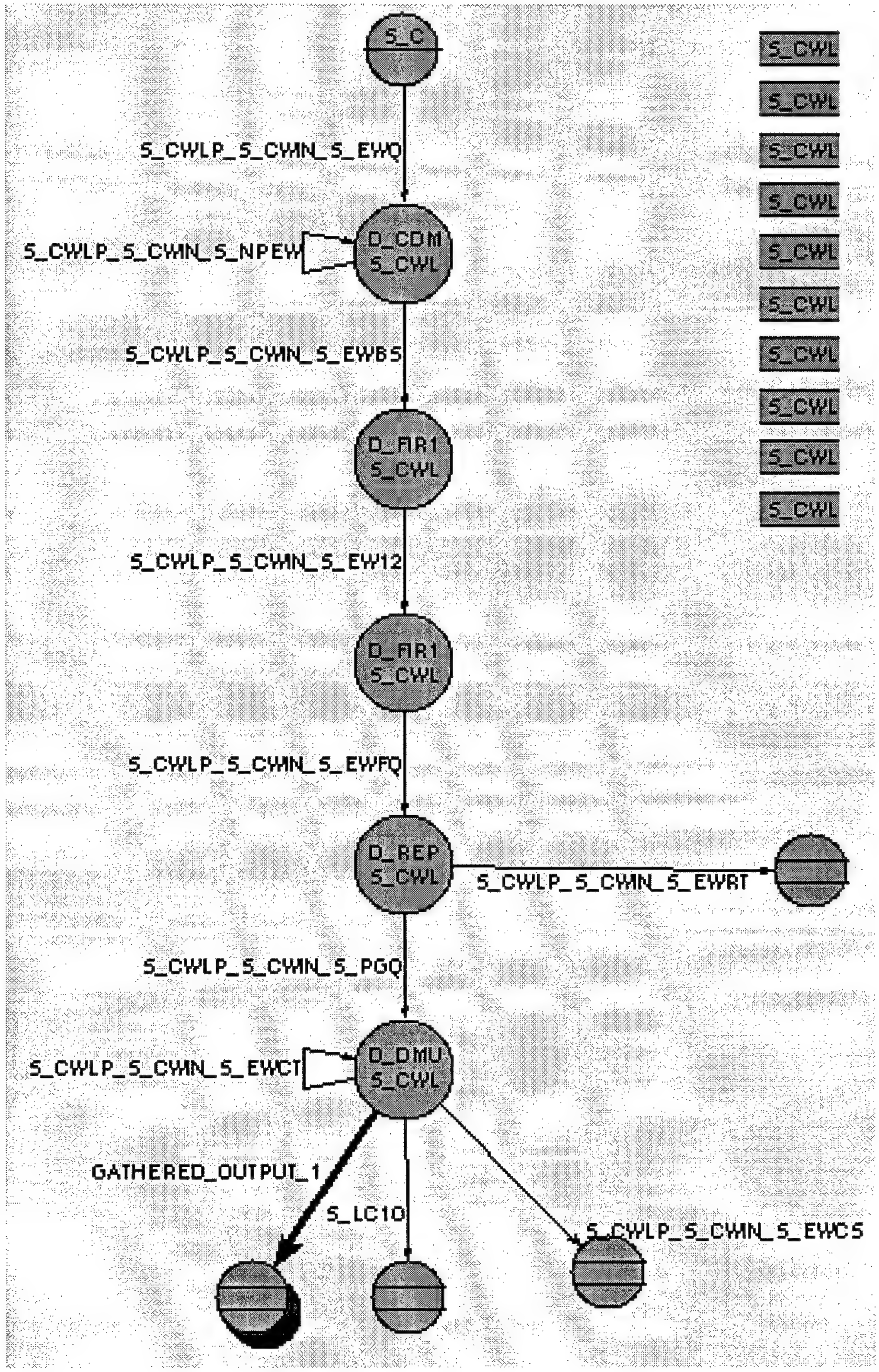


Figure 47. Partition P_CWLIN_4

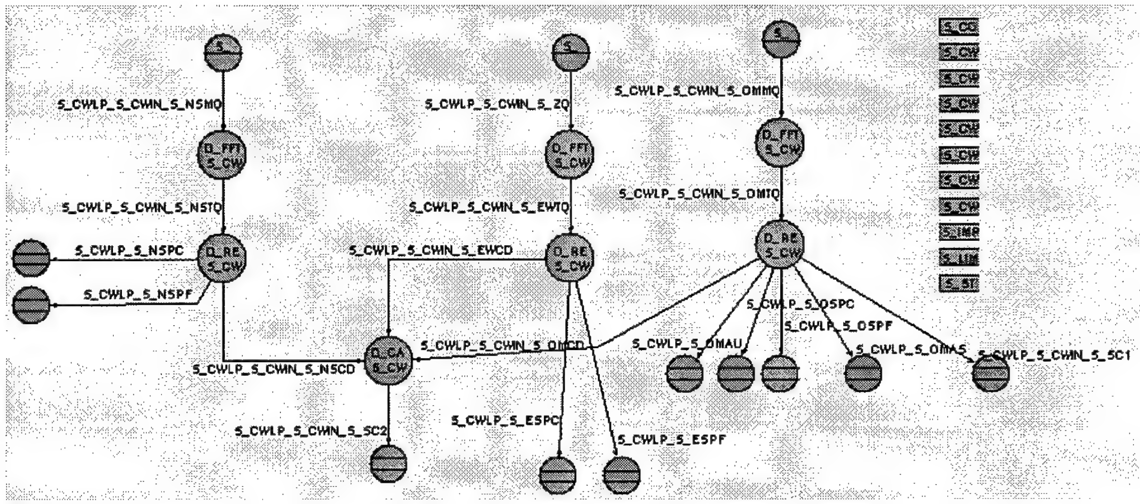


Figure 48. Partition P_CWLIN_5

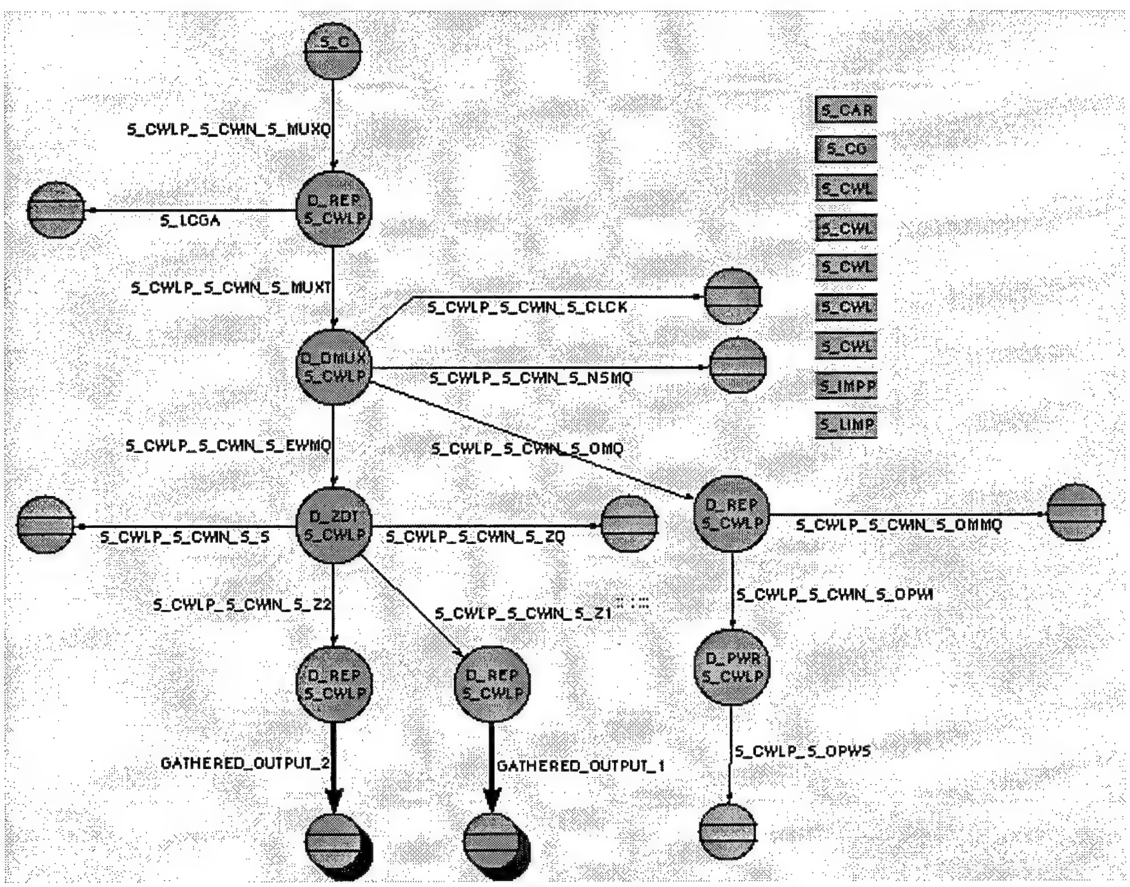


Figure 49. Partition P_CWLIN_5B

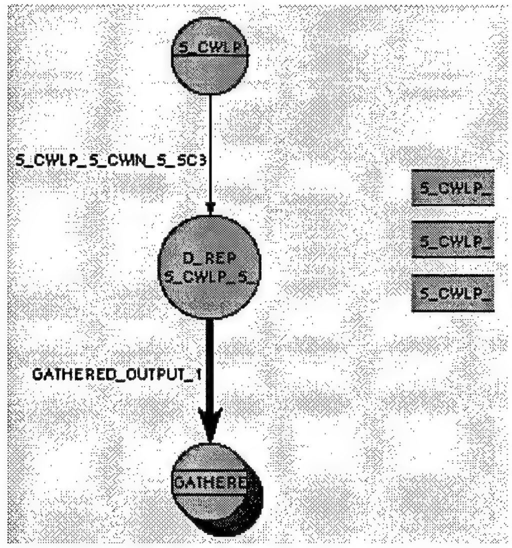


Figure 50. Partition P_CWLIN_6

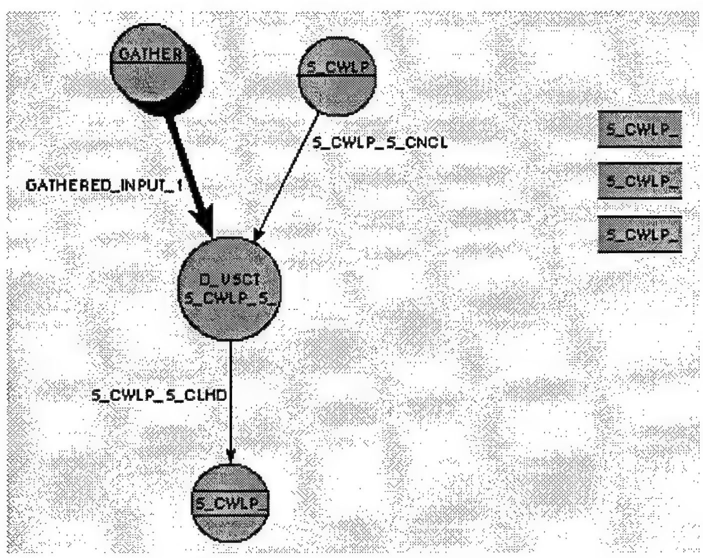


Figure 51. Partition P_CWLIN_7

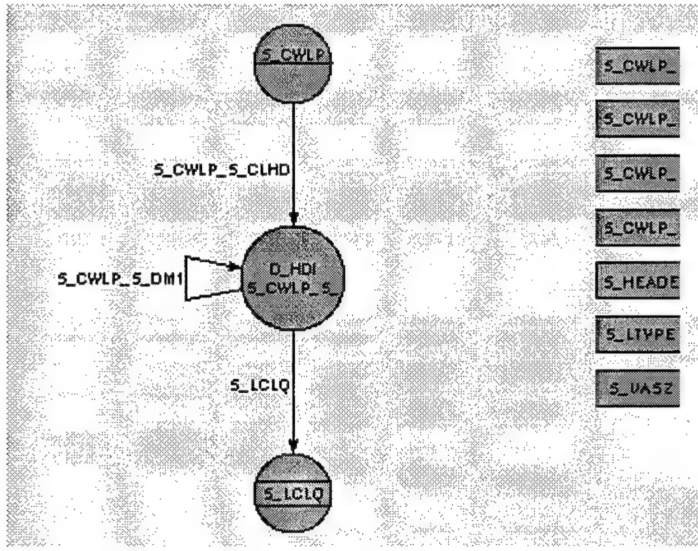


Figure 52. Partition P_CWLIN_7A

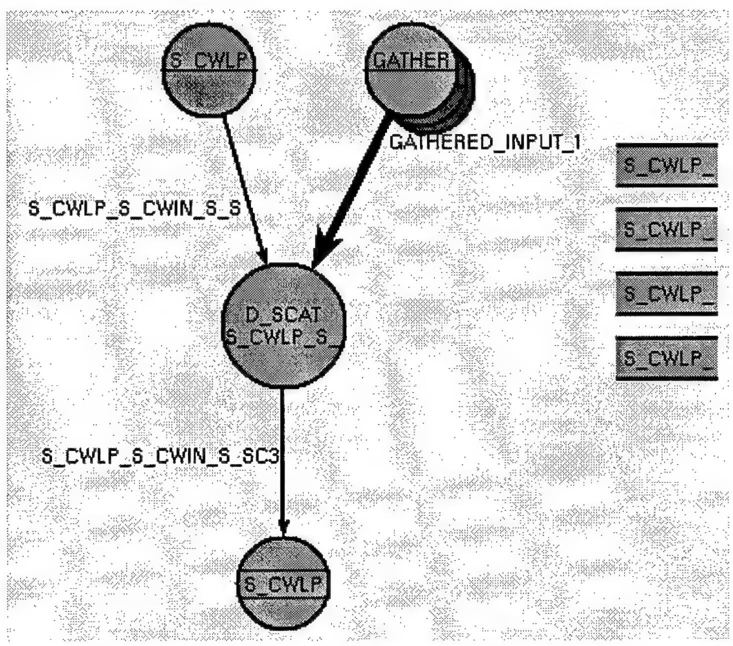


Figure 53. Partition P_CWLIN_8

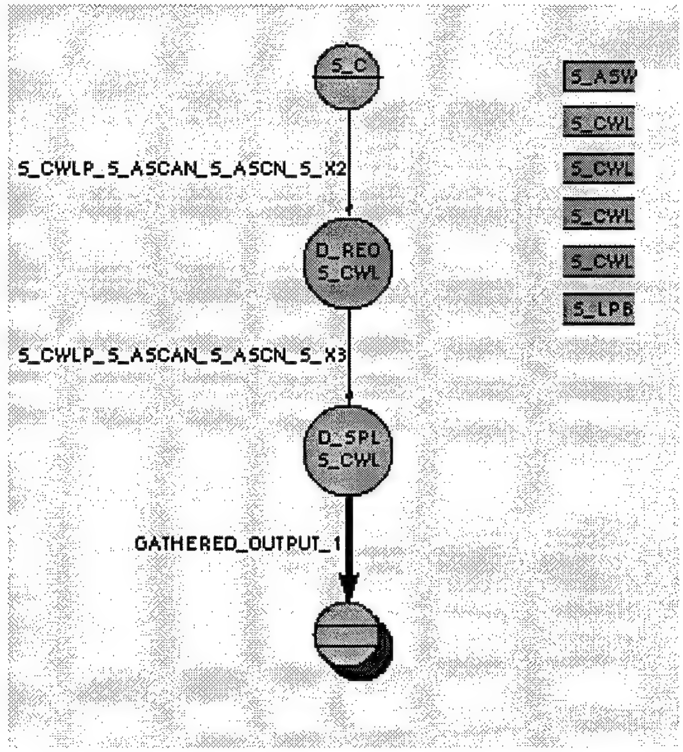


Figure 54. Partition P_CWL_1

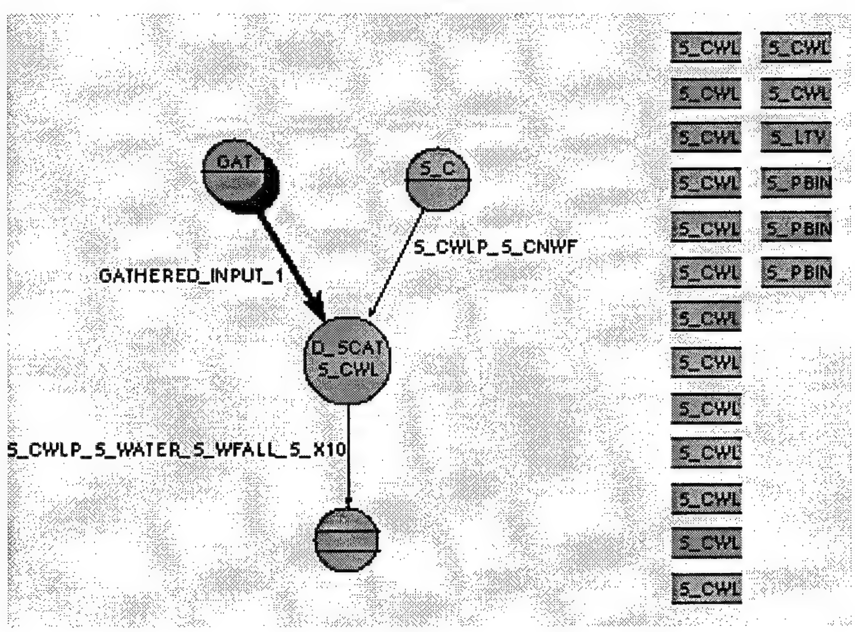


Figure 55. Partition P_CWL_1A

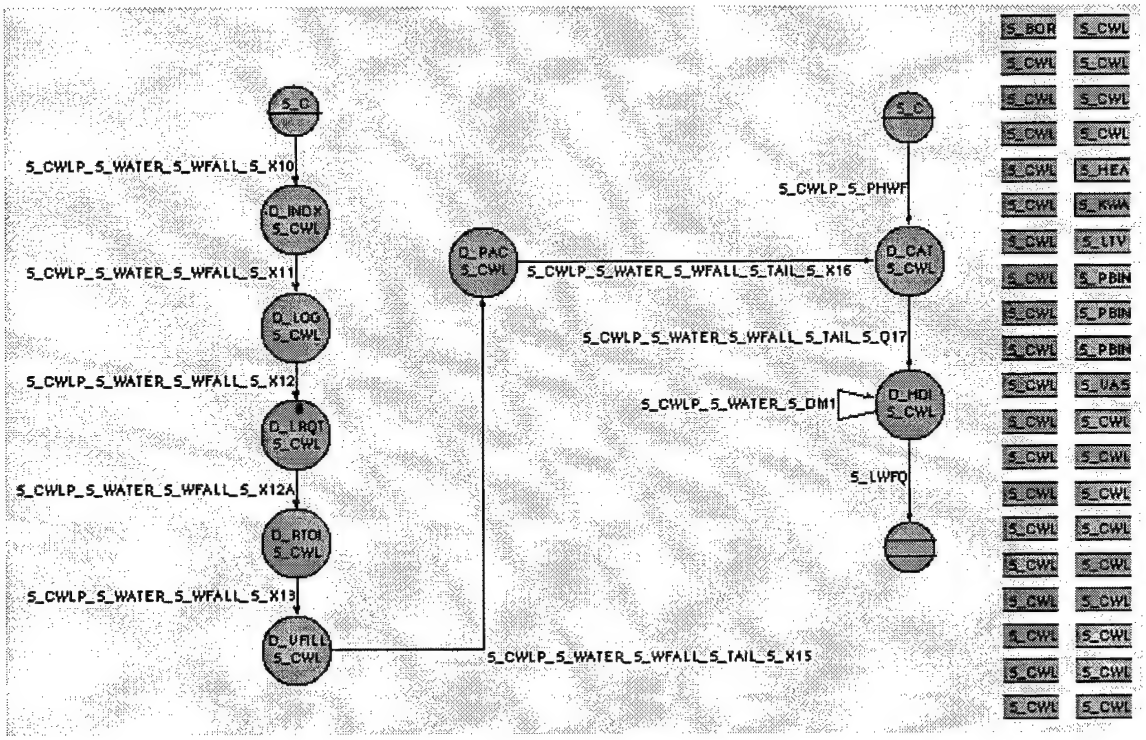


Figure 56. Partition P_CWL_1B

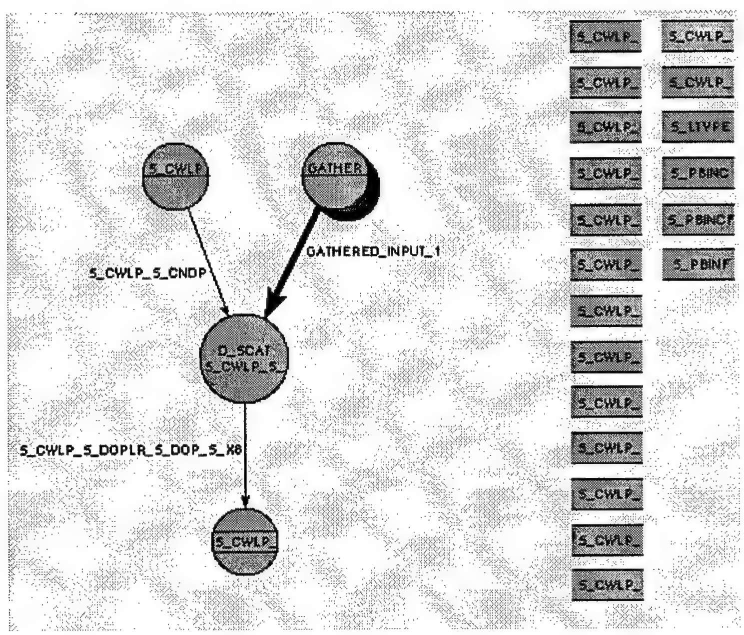


Figure 57. Partition P_CWL_1C

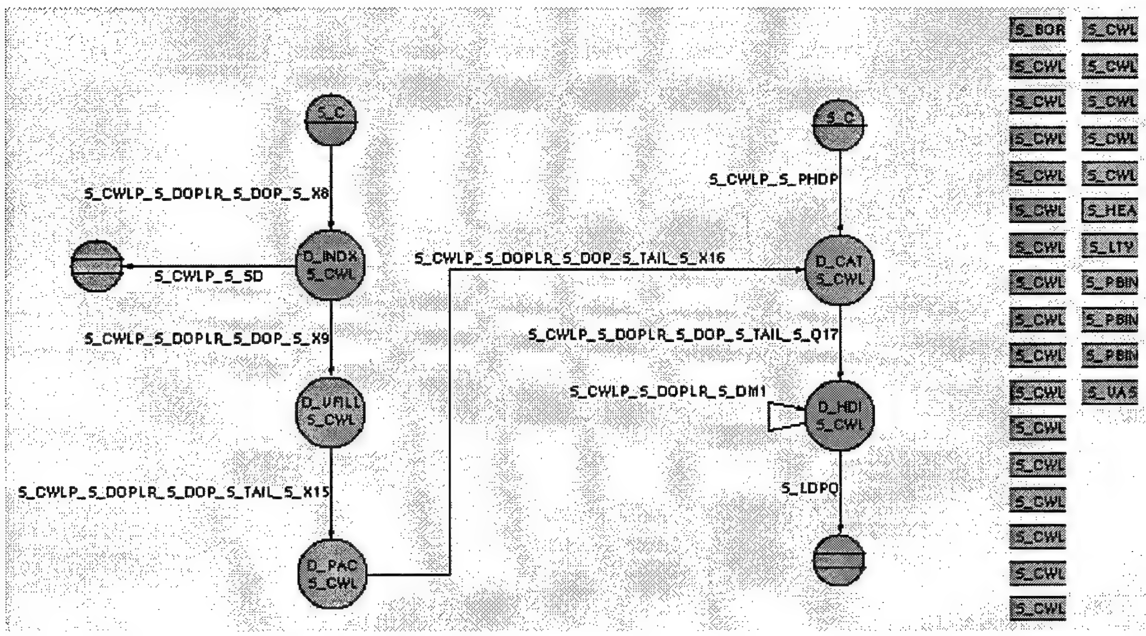


Figure 58. Partition P_CWL_1D

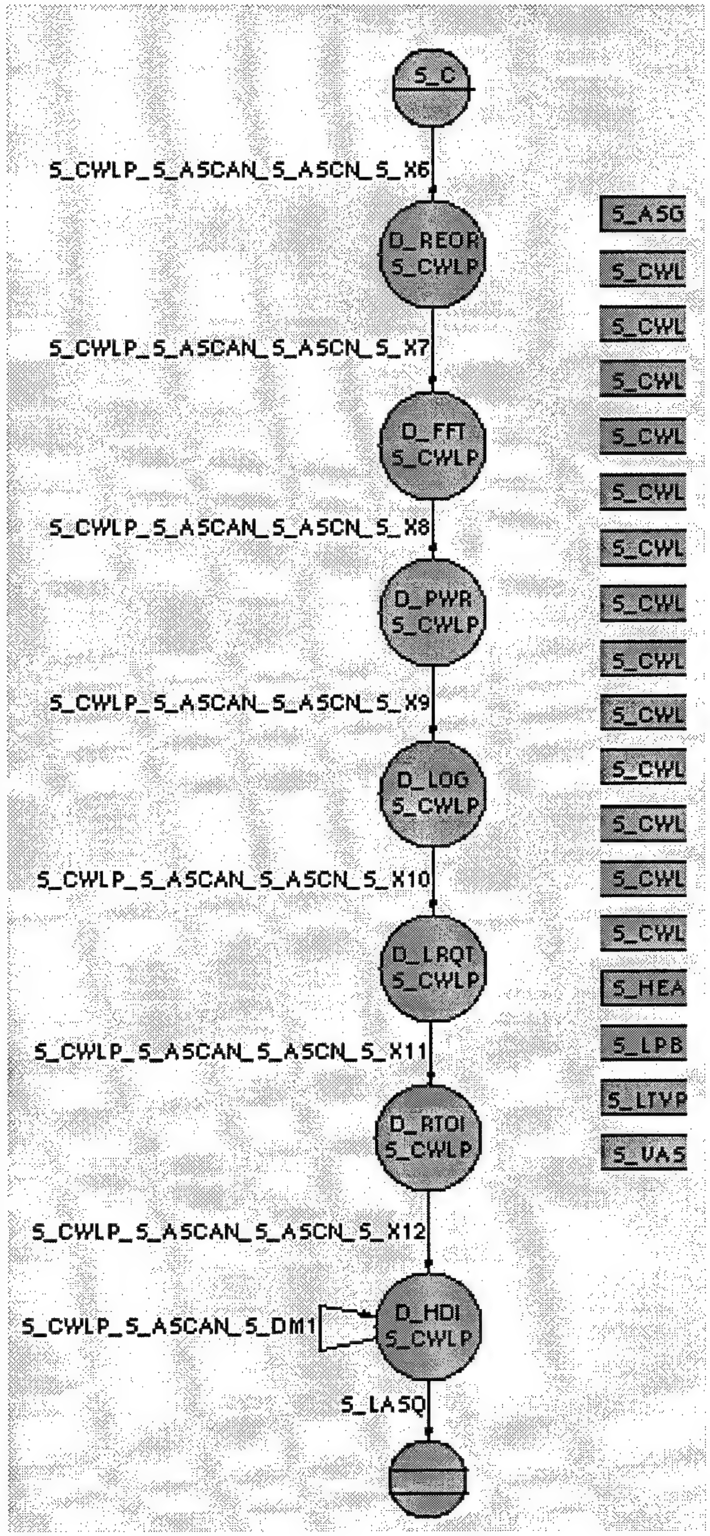


Figure 59. Partition P_CWL_1E

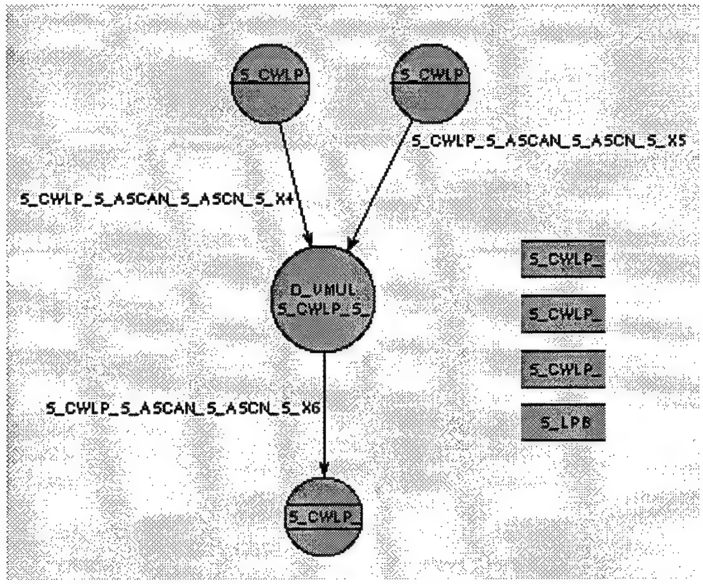


Figure 60. Partition P_CWL_1F

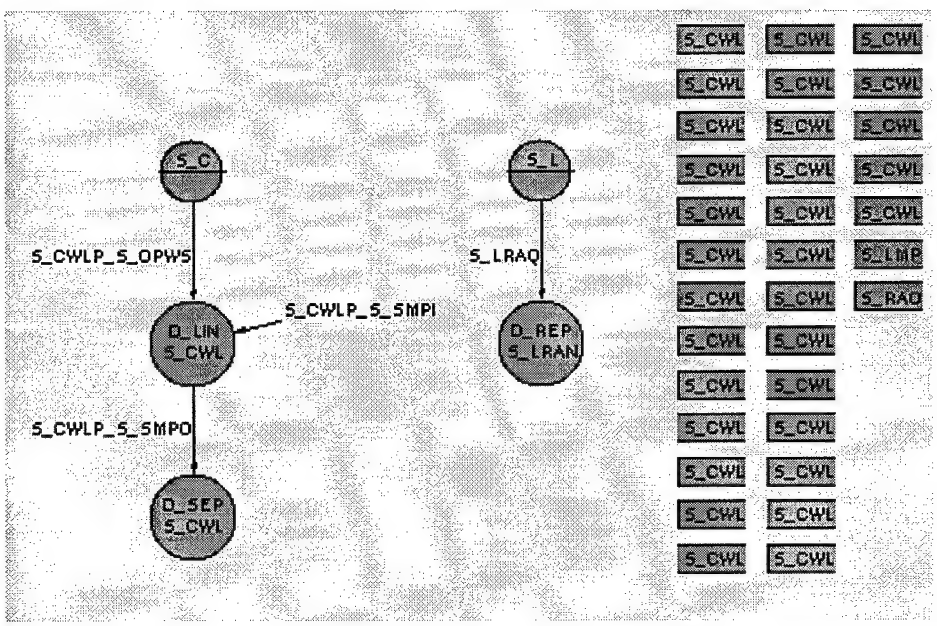


Figure 61. Partition P_CWL_1K

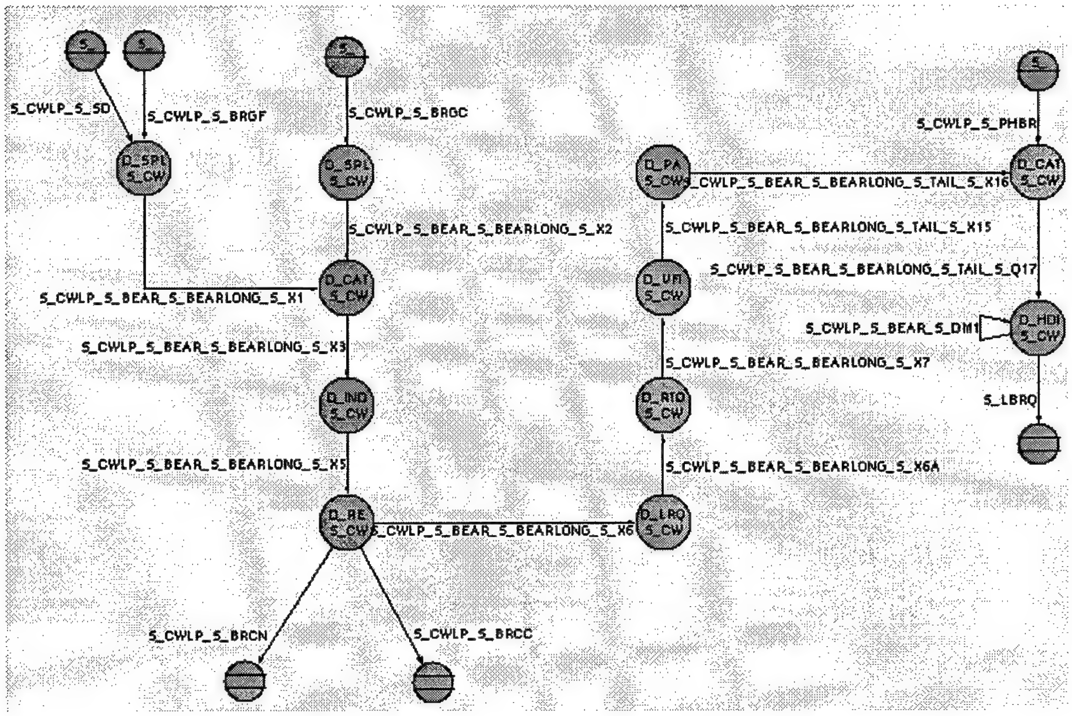


Figure 62. Partition P_CWL_1L

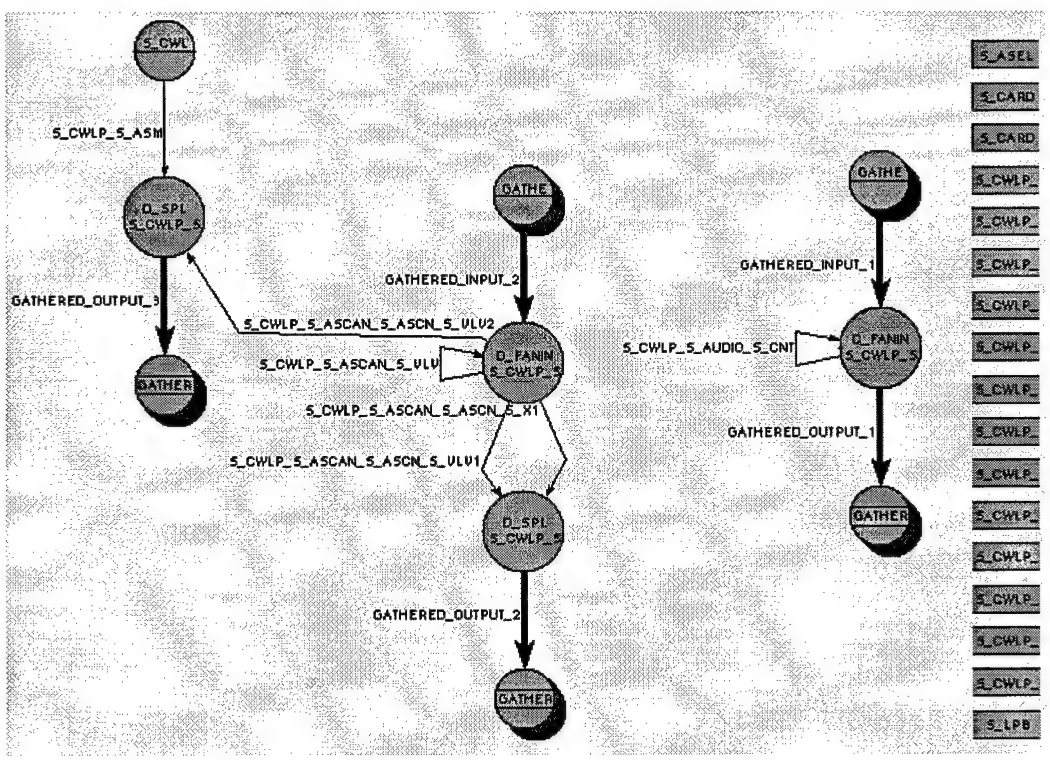


Figure 63. Partition P_CWL_2

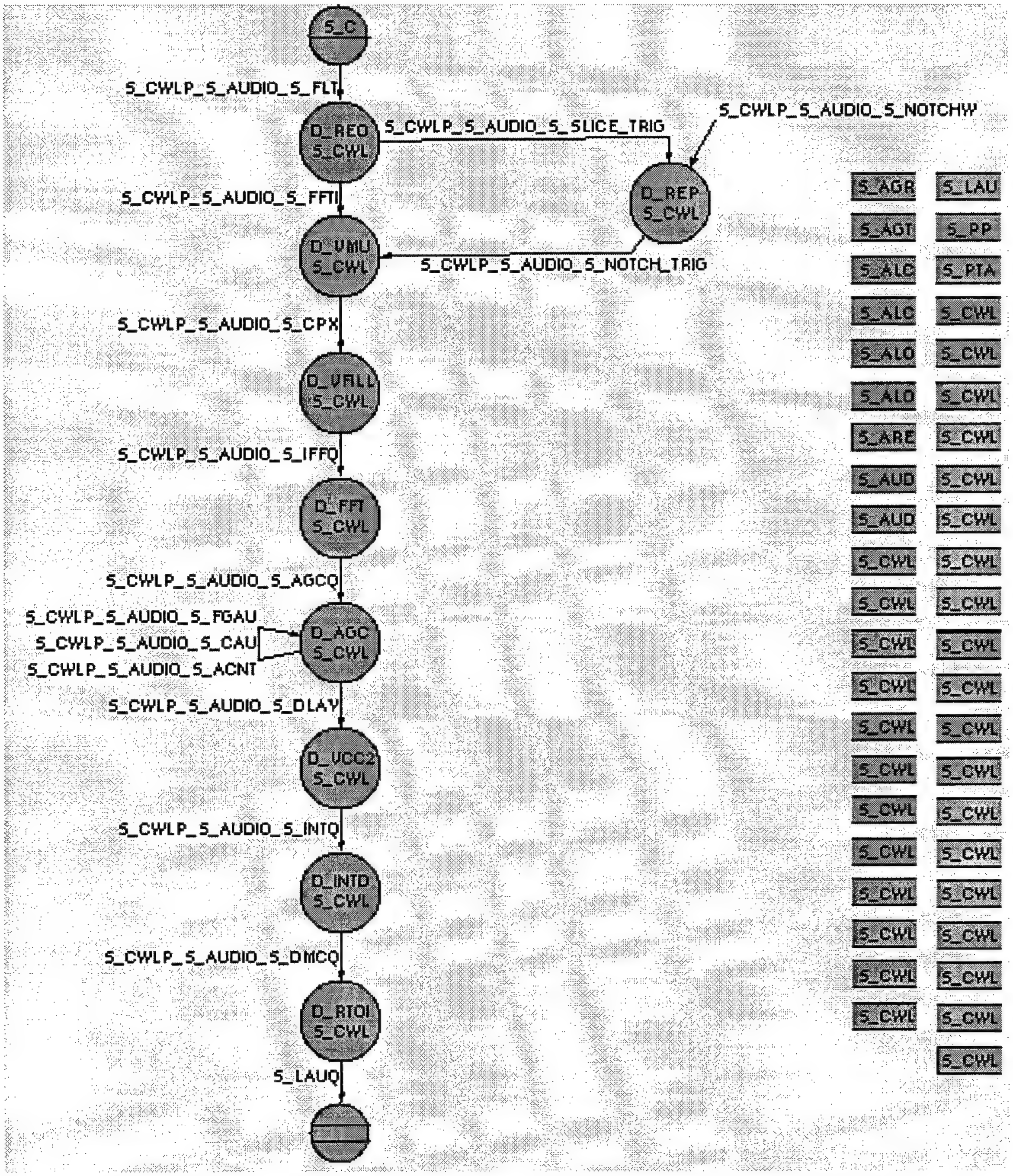


Figure 64. Partition P_CWL_2B

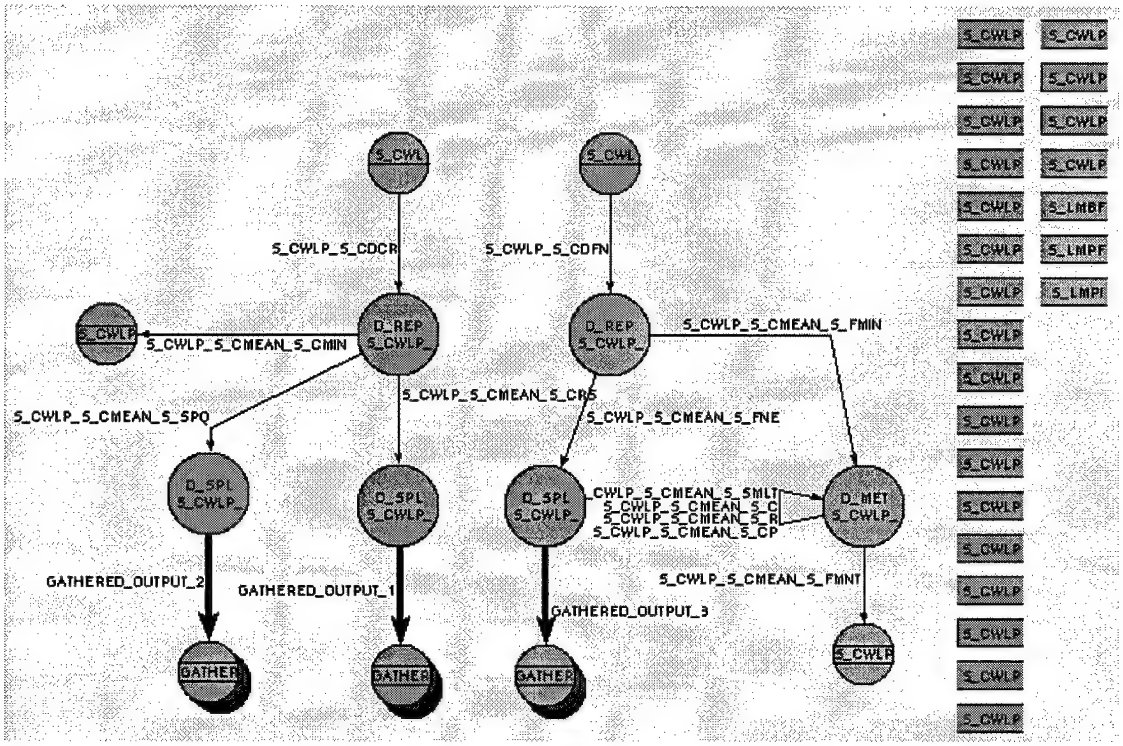


Figure 65. Partition P_CWL_3

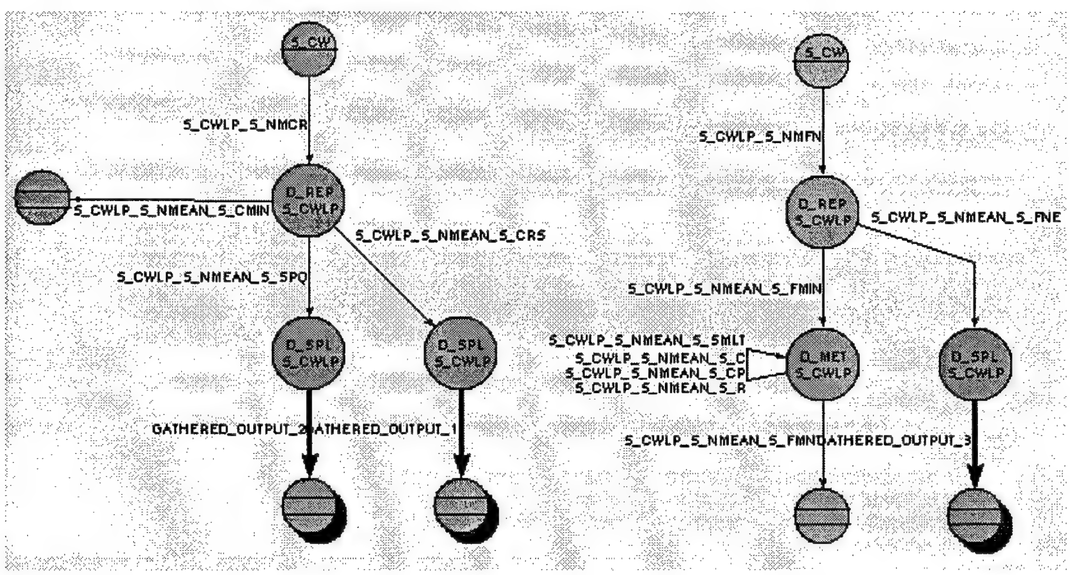


Figure 66. Partition P_CWL_3A

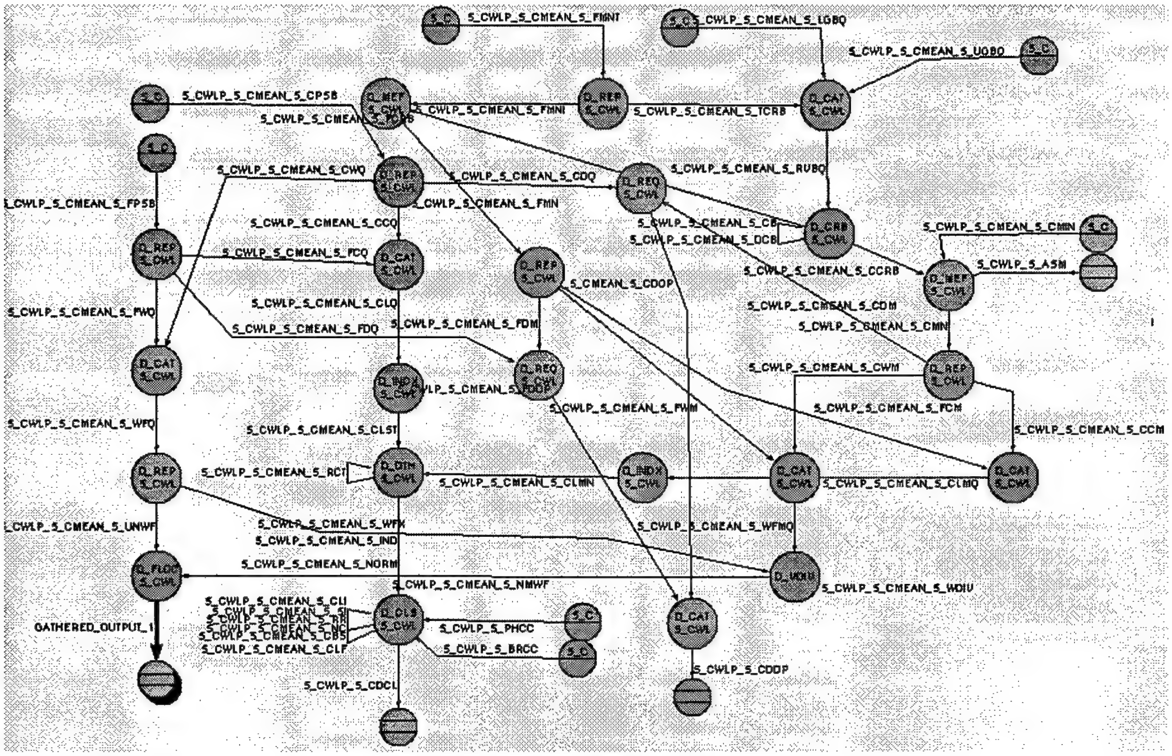


Figure 67. Partition P_CWL_3B

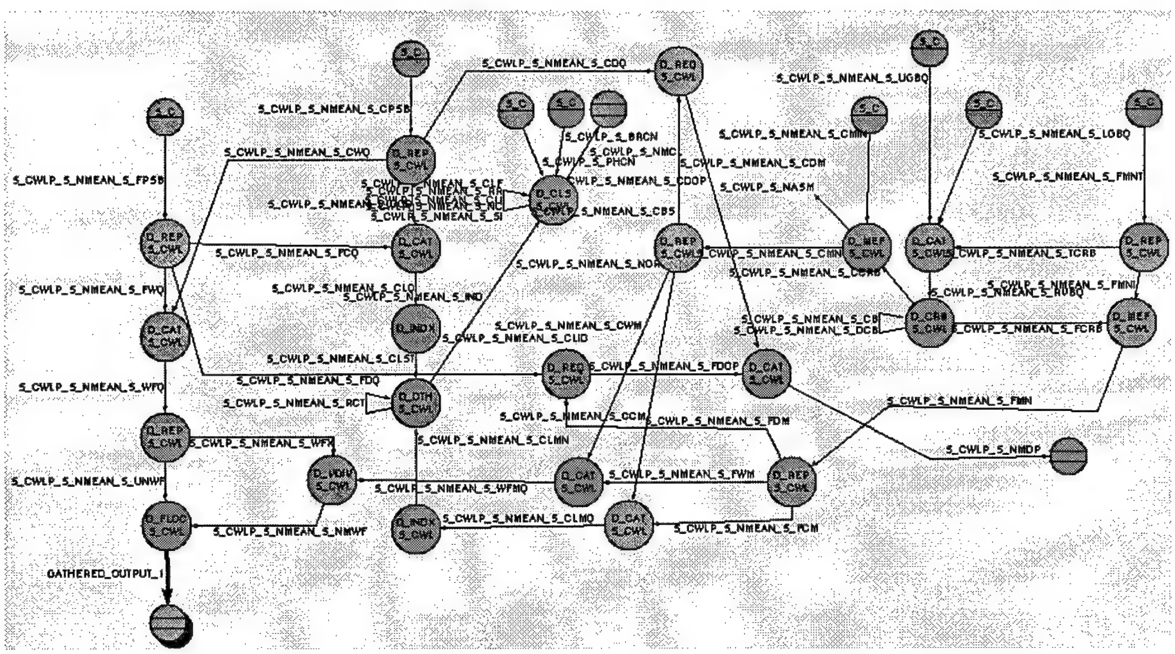


Figure 68. Partition P_CWL_3C

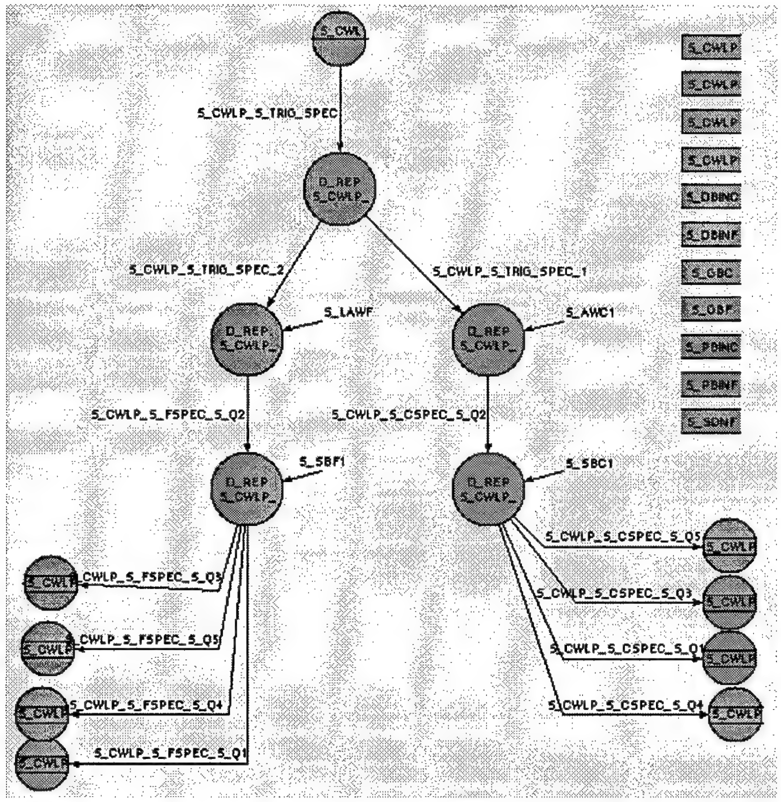


Figure 69. Partition P_CWL_4

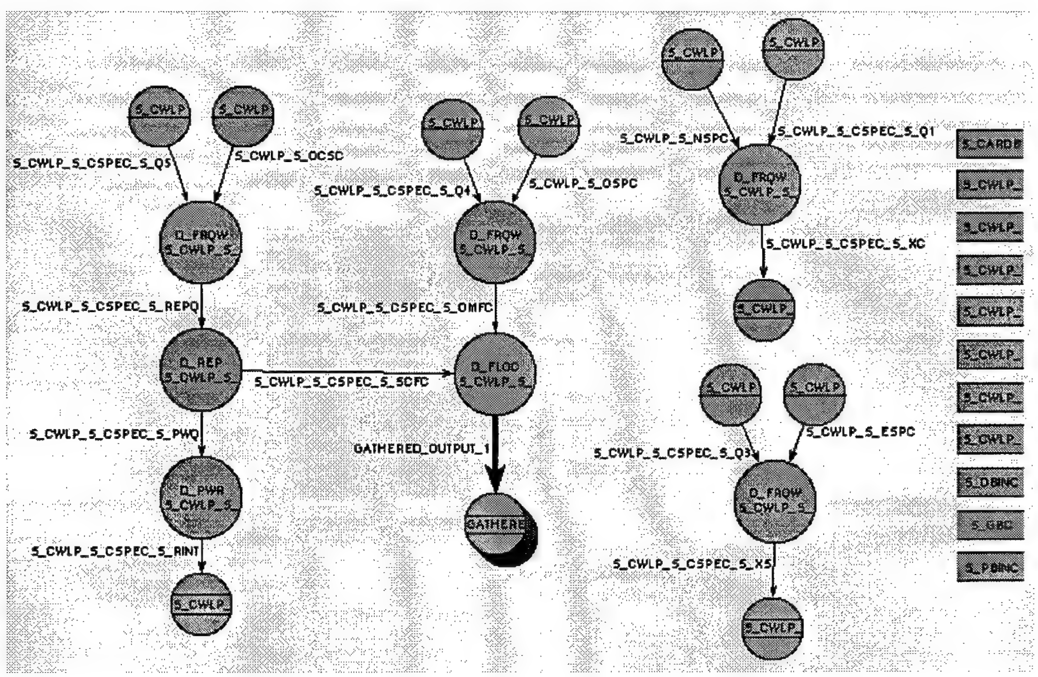


Figure 70. Partition P_CWL_4A

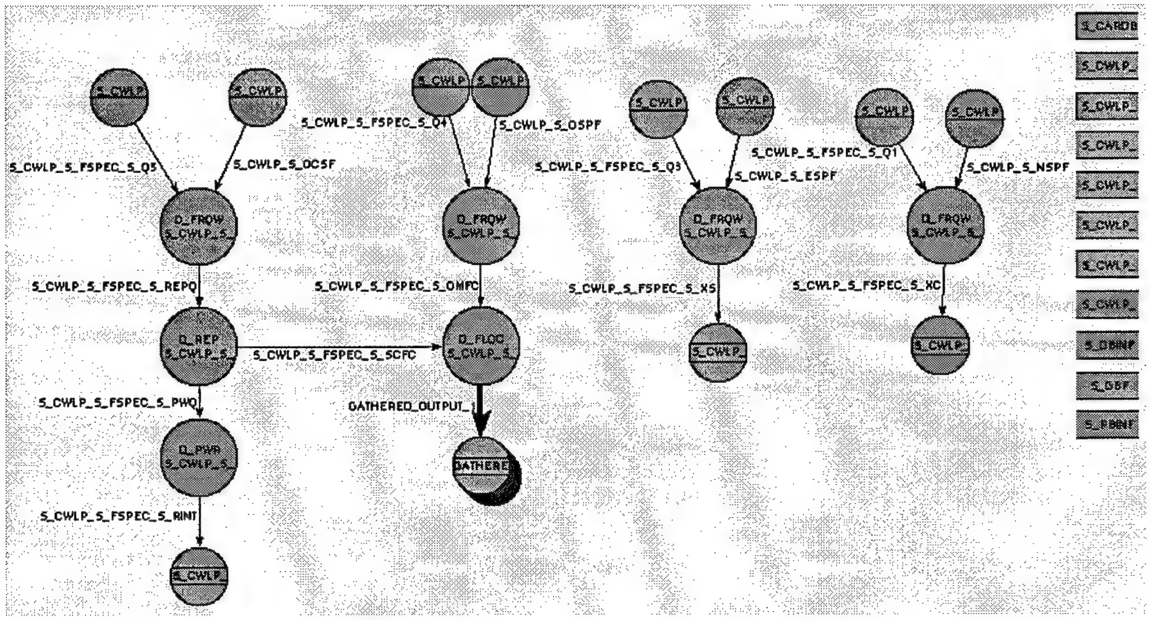


Figure 71. Partition P_CWL_4B

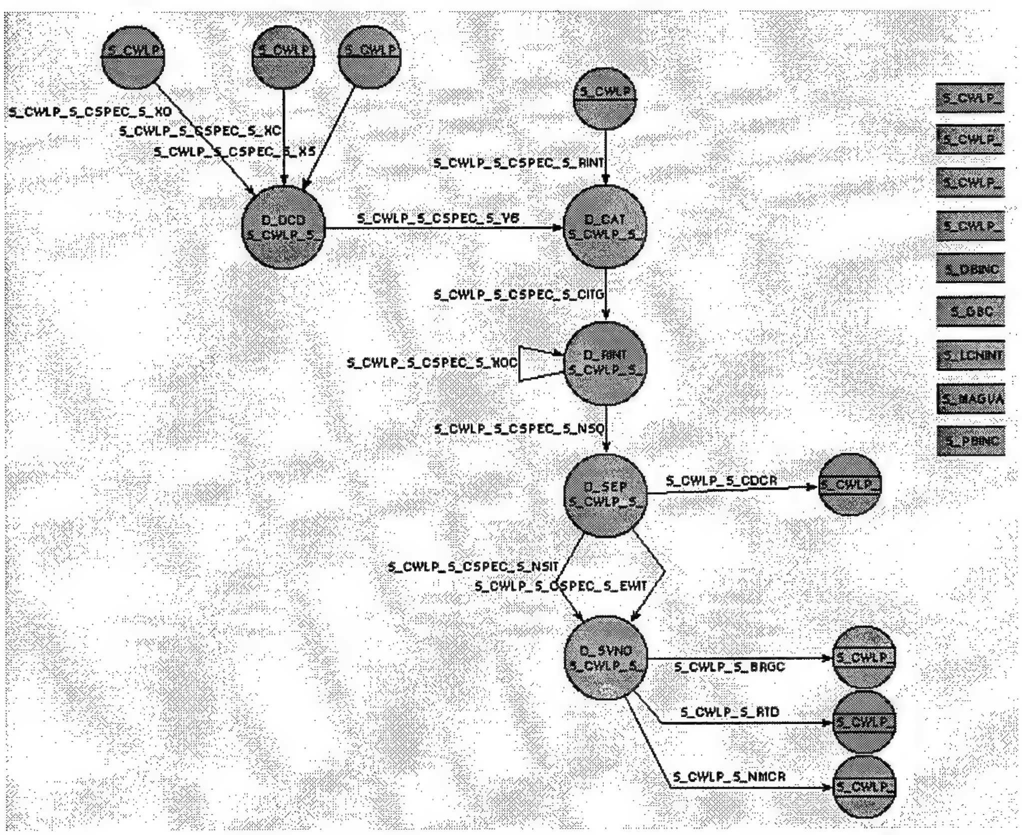


Figure 72. Partition P_CWL_4C

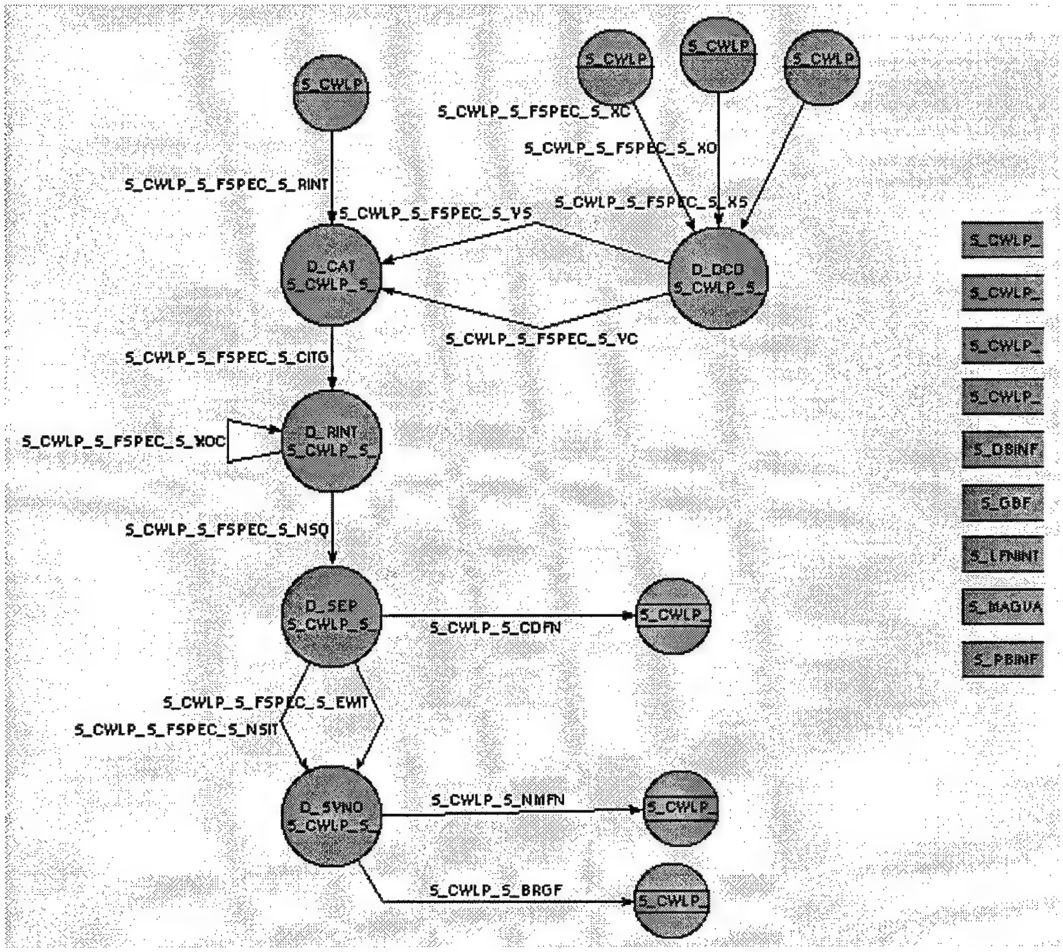


Figure 73. Partition P_CWL_4D

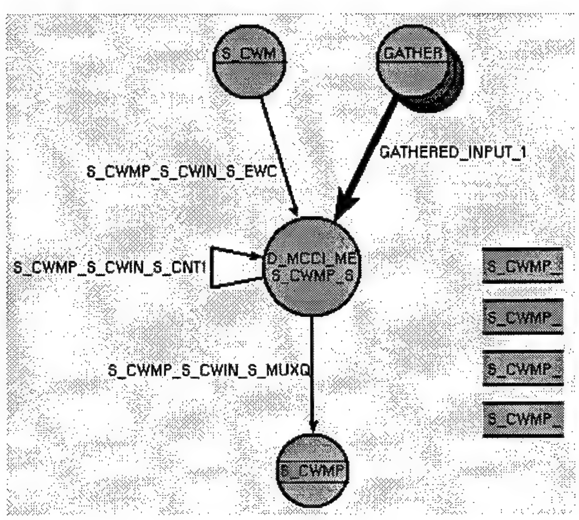


Figure 74. Partition P_CWMIN_1

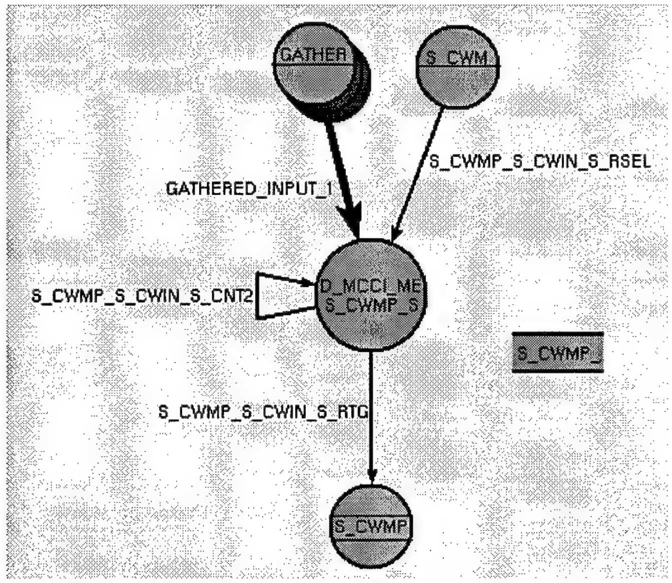


Figure 75. Partition P_CWMIN_2

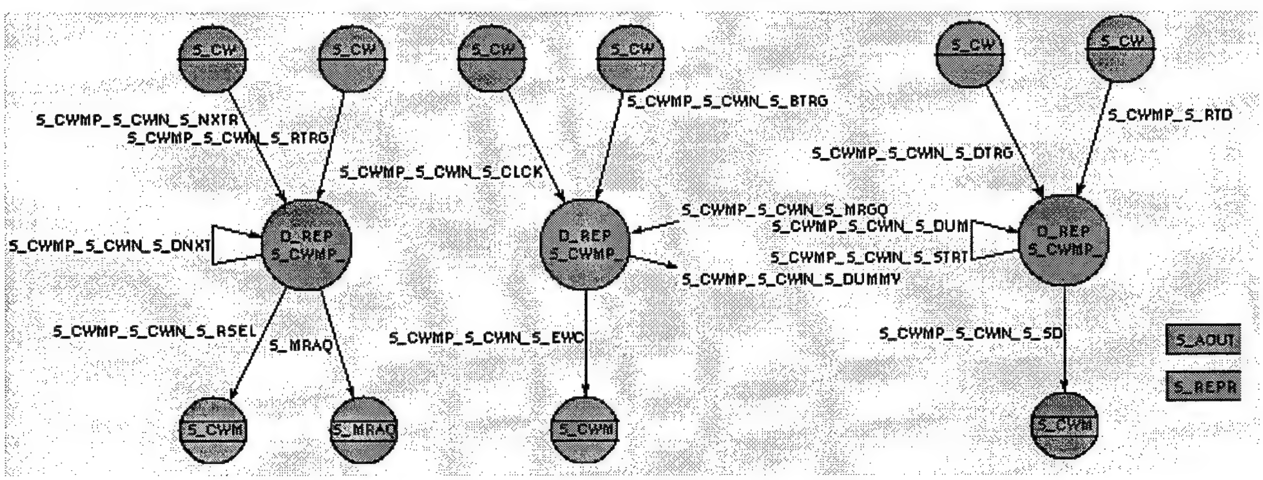


Figure 76. Partition P_CWMIN_3

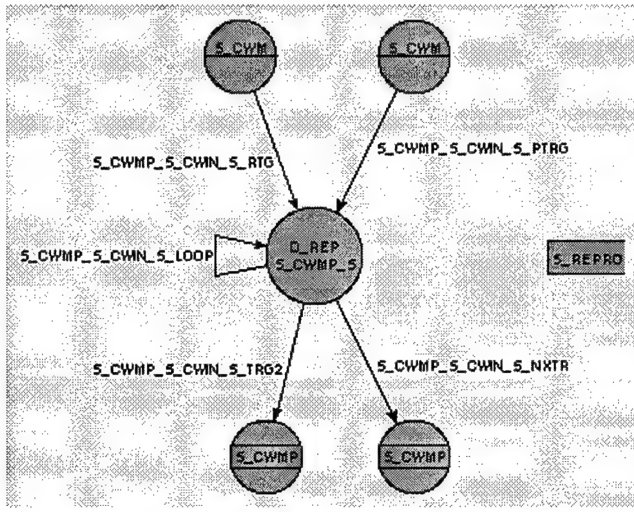


Figure 77. Partition P_CWMIN_3B

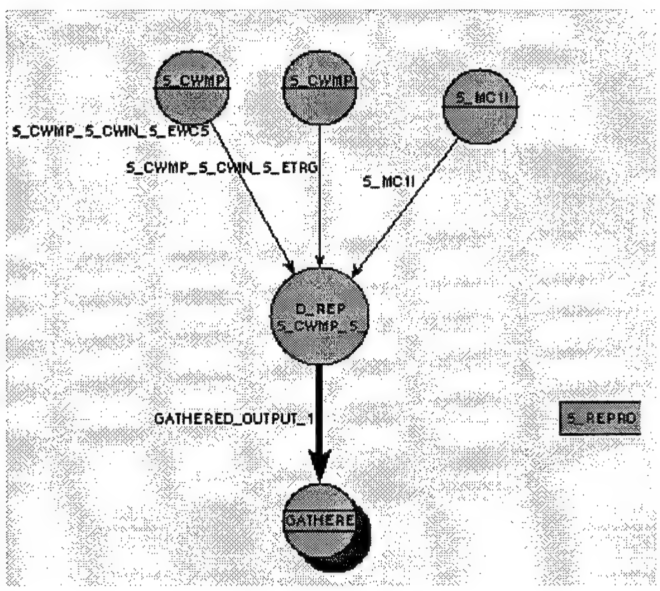


Figure 78. Partition P_CWMIN_3C

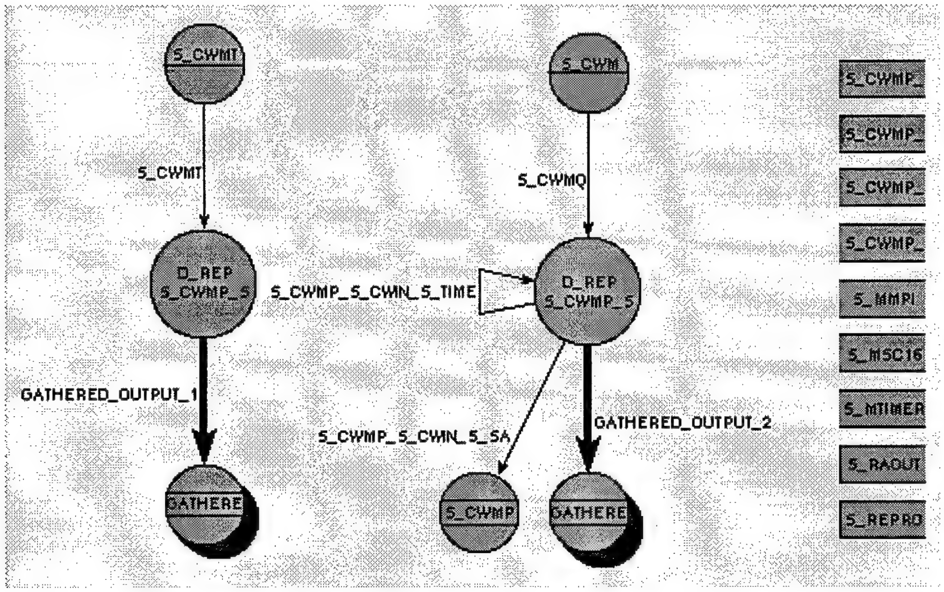


Figure 79. Partition P_CWMIN_3D

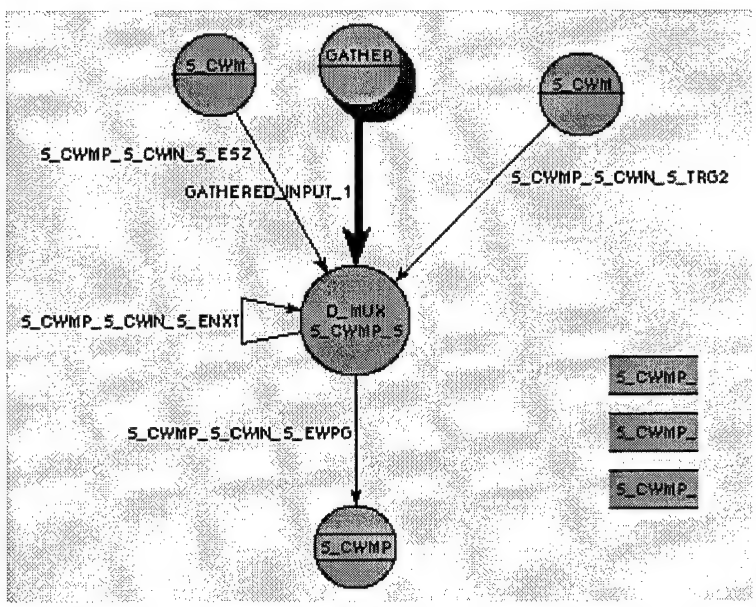


Figure 80. Partition P_CWMIN_3E

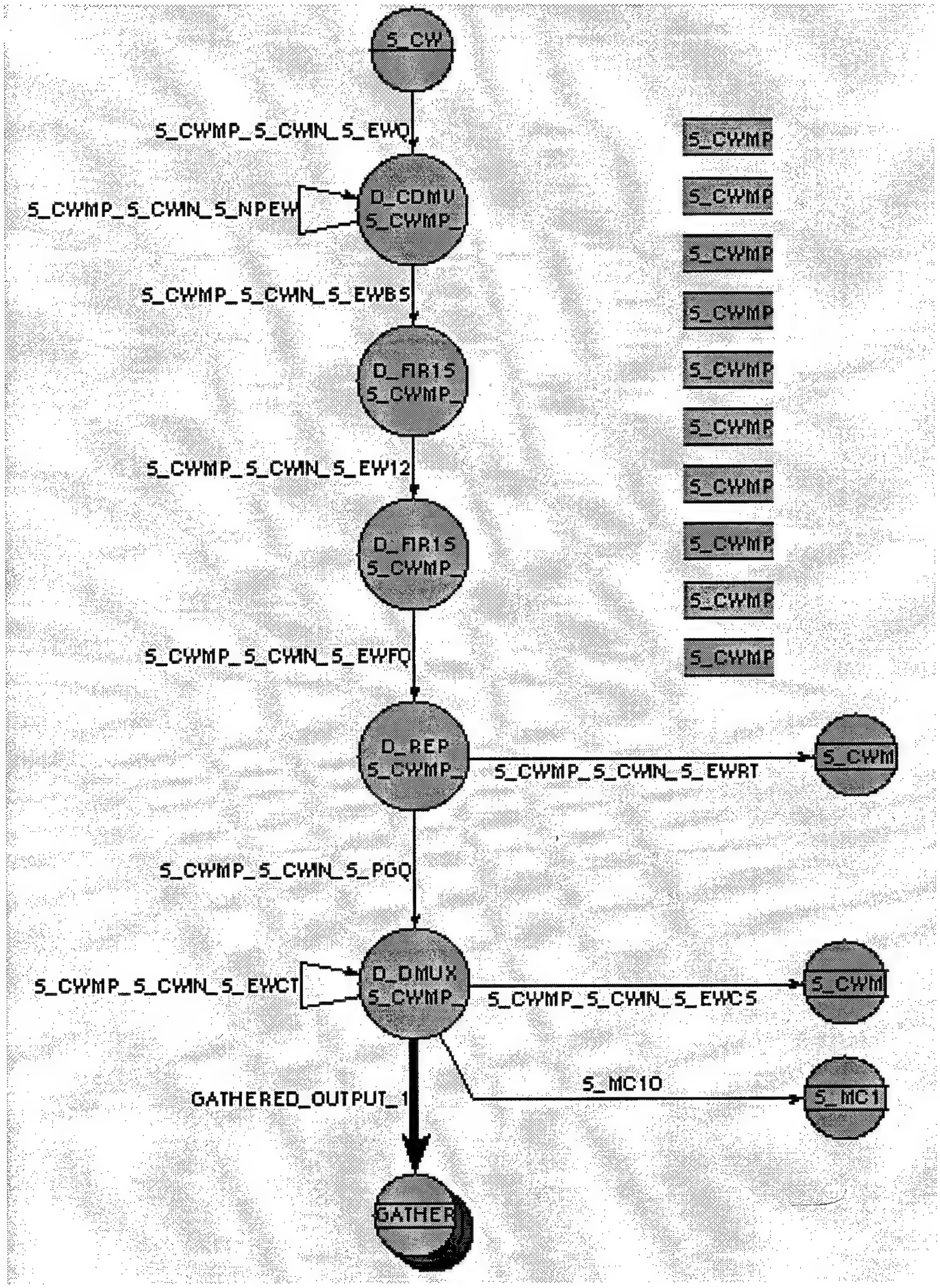


Figure 81. Partition P_CWMIN_4

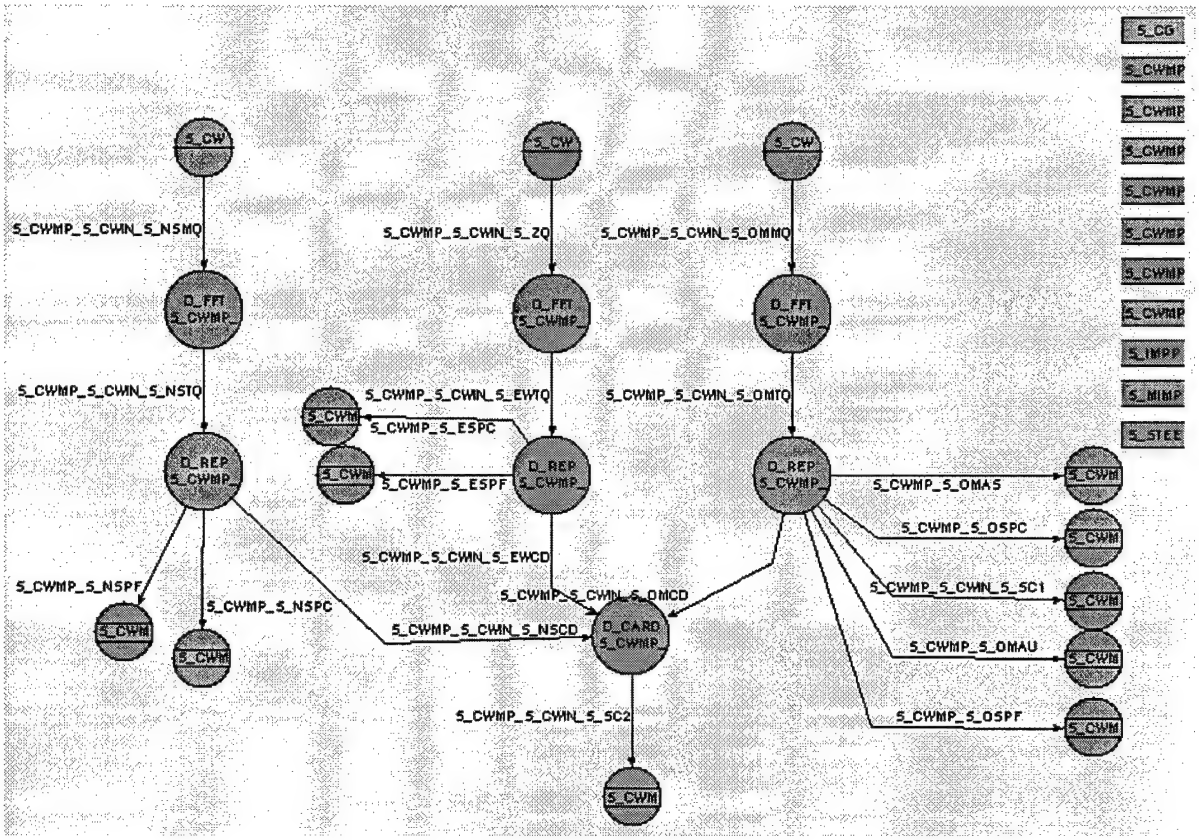


Figure 82. Partition P_CWMIN_5

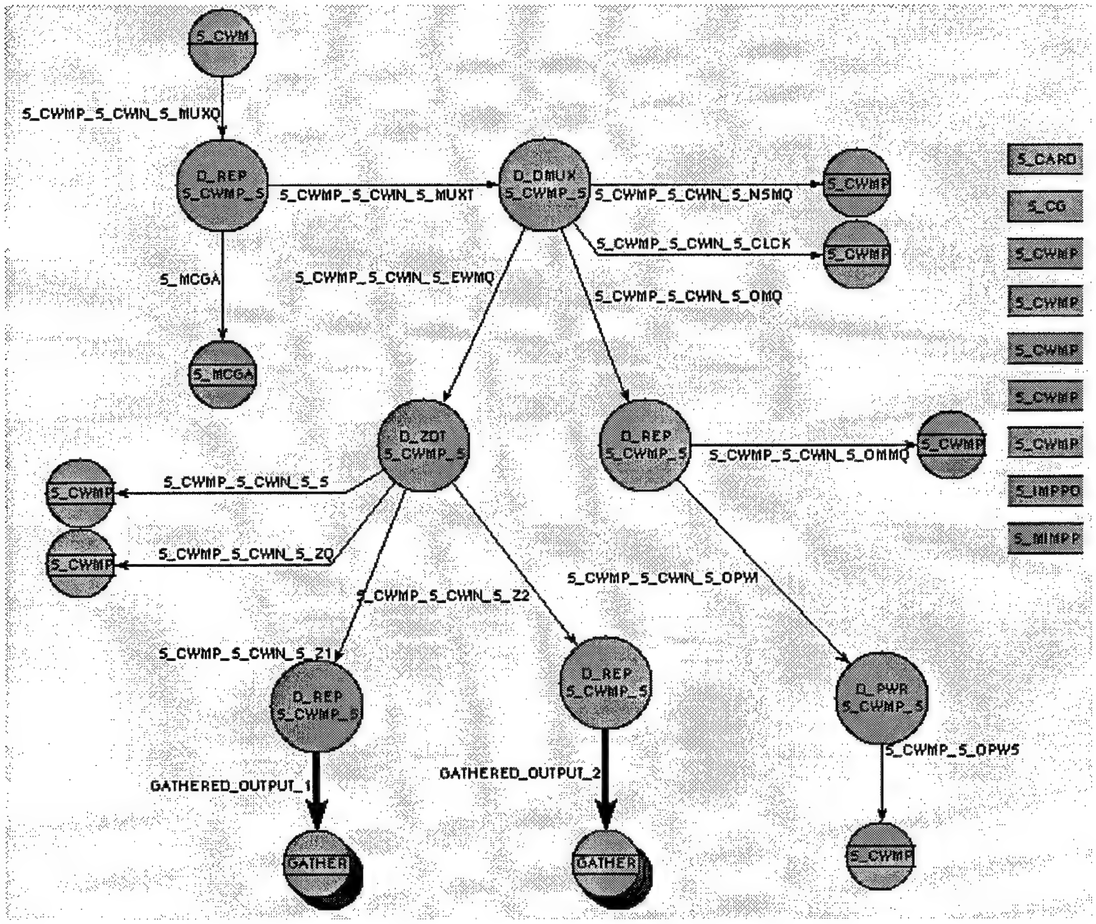


Figure 83. Partition P_CWMIN_5B

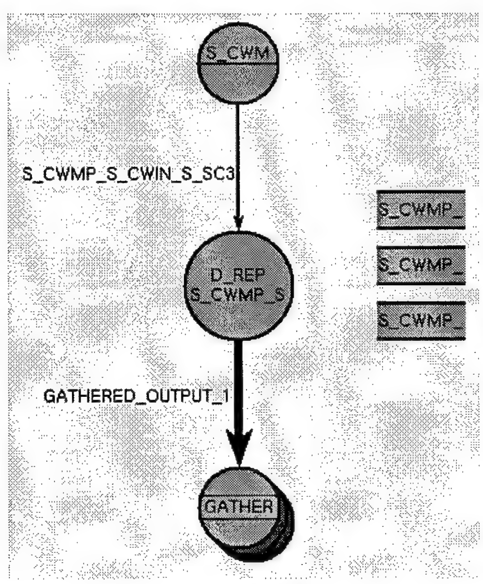


Figure 84. Partition P_CWMIN_6

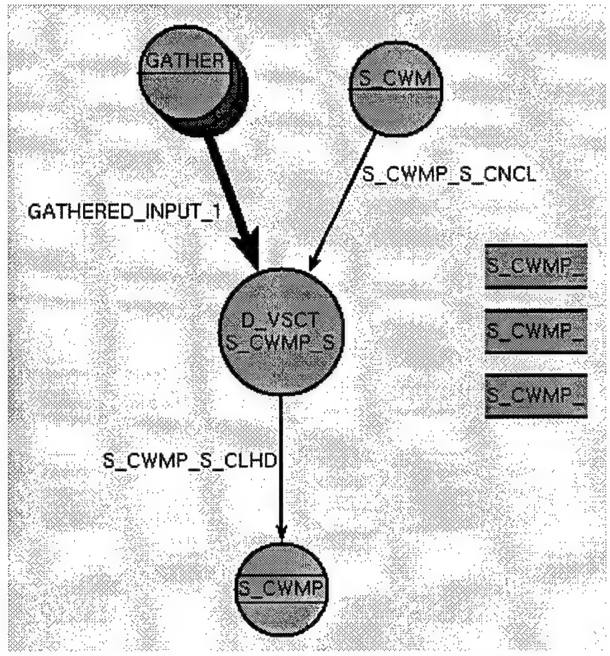


Figure 85. Partition P_CWMIN_7

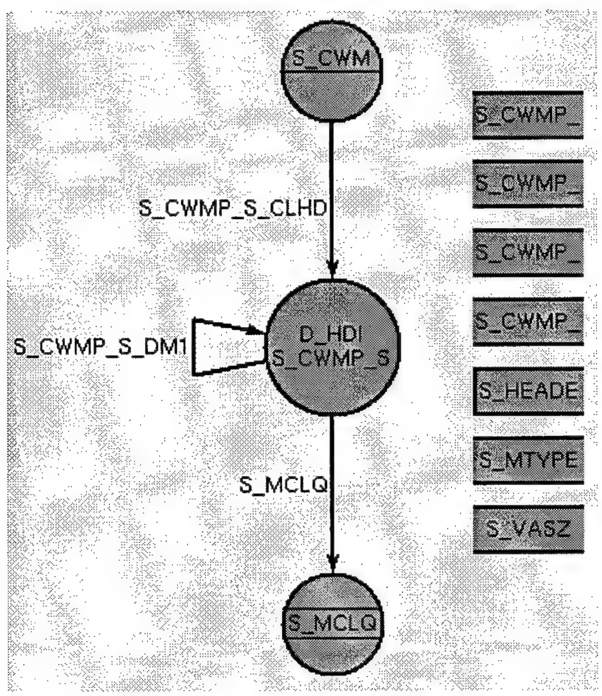


Figure 86. Partition P_CWMIN_7A

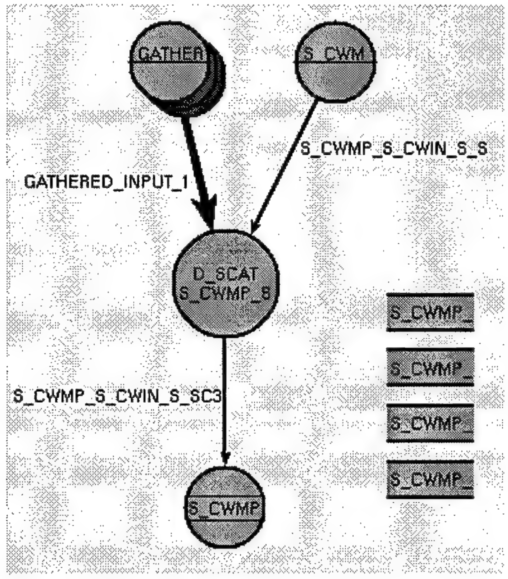


Figure 87. Partition P_CWMIN_8

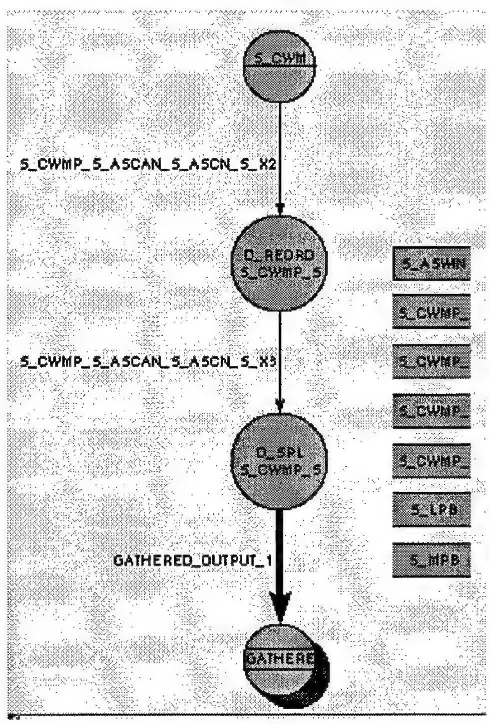


Figure 88. Partition P_CWM_1

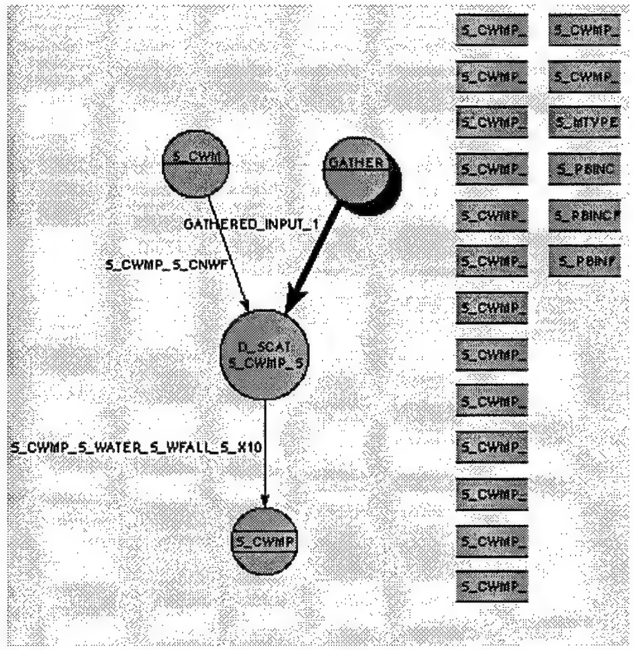


Figure 89. Partition P_CWM_1A

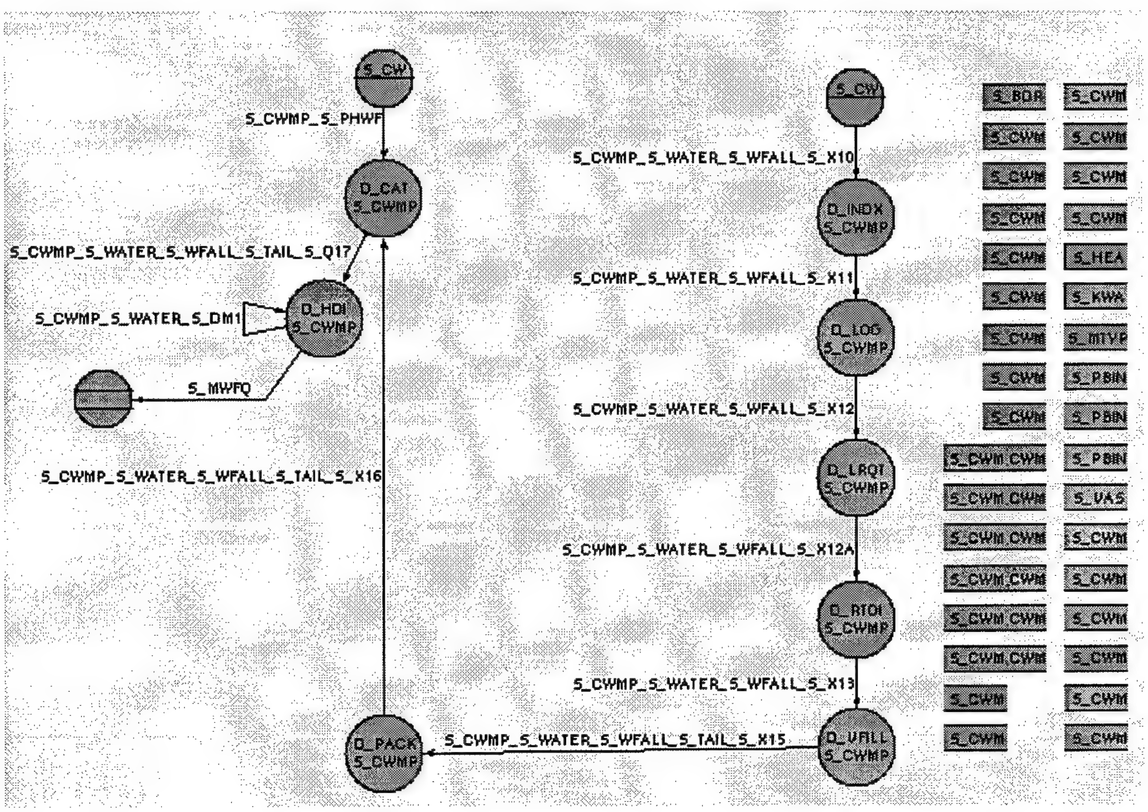


Figure 90. Partition P_CWM_1B

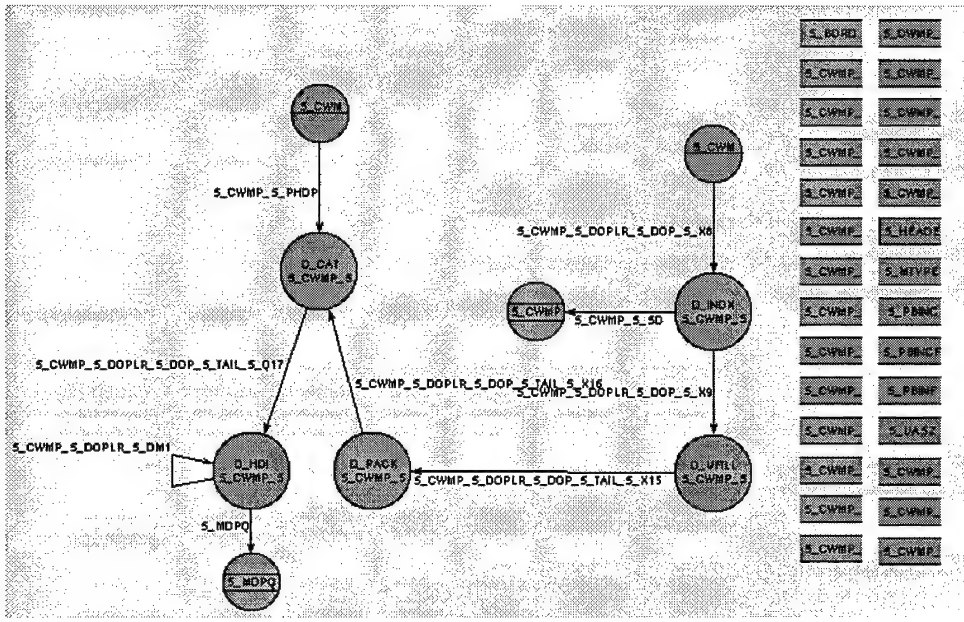


Figure 91. Partition P_CWM_1C

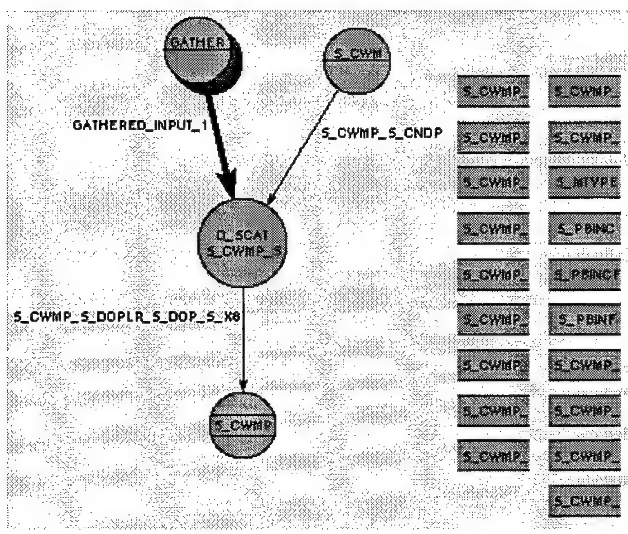


Figure 92. Partition P_CWM_1D

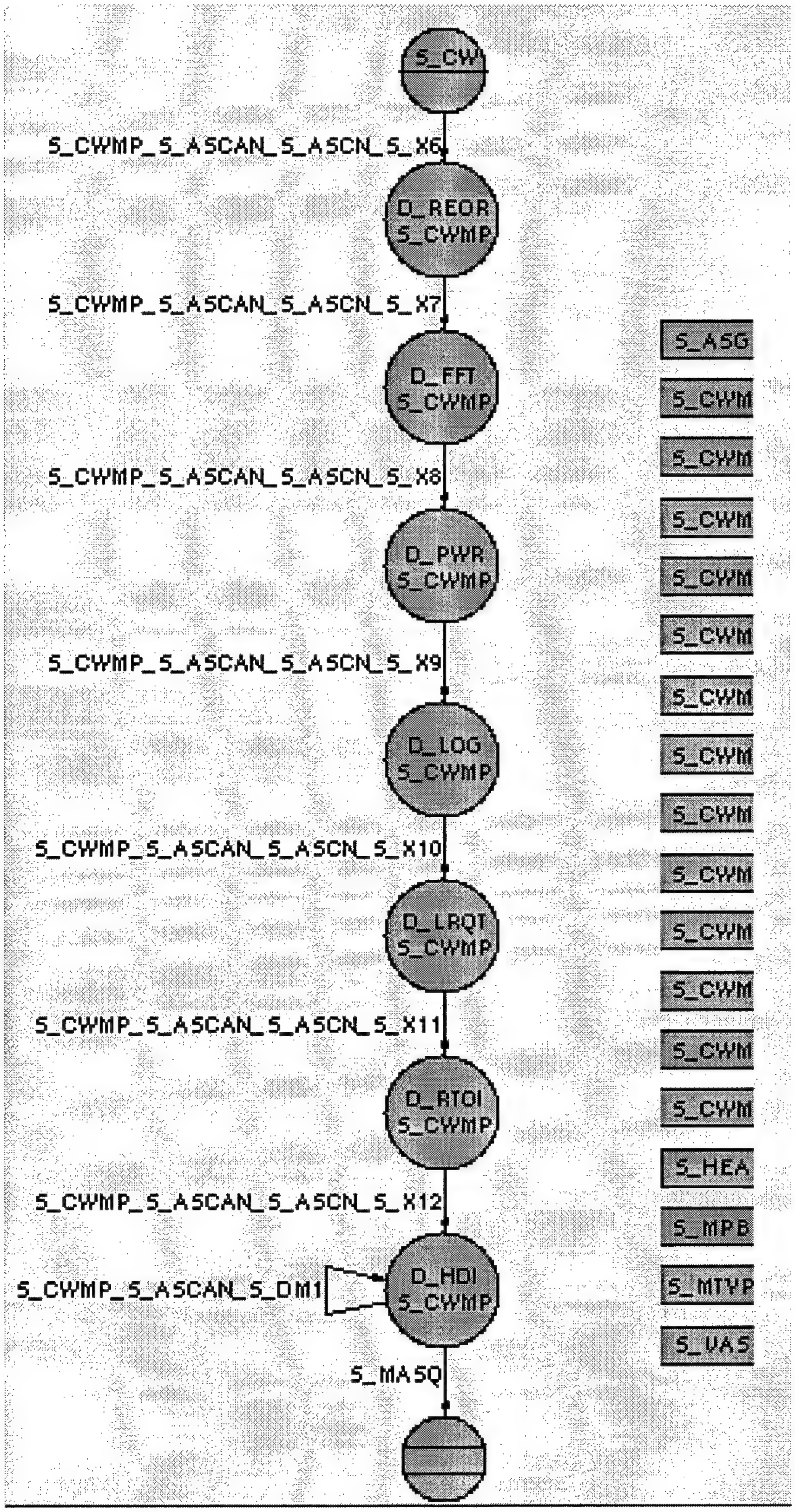


Figure 93. Partition P_CWM_1E

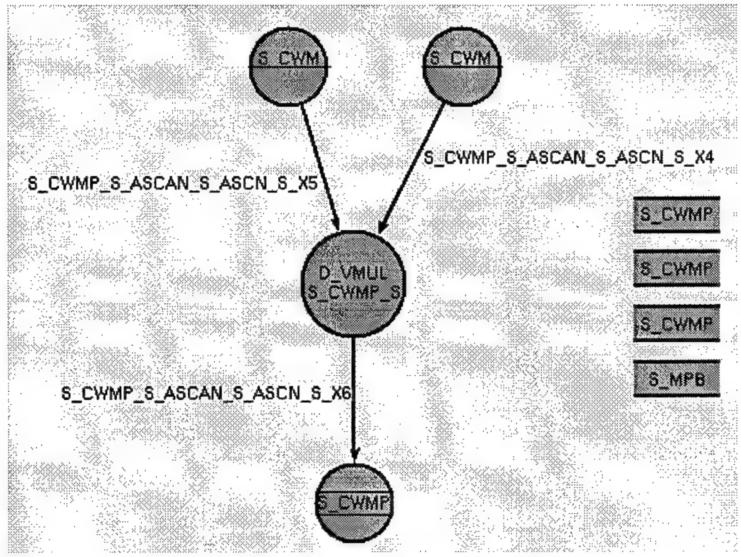


Figure 94. Partition P_CWM_1F

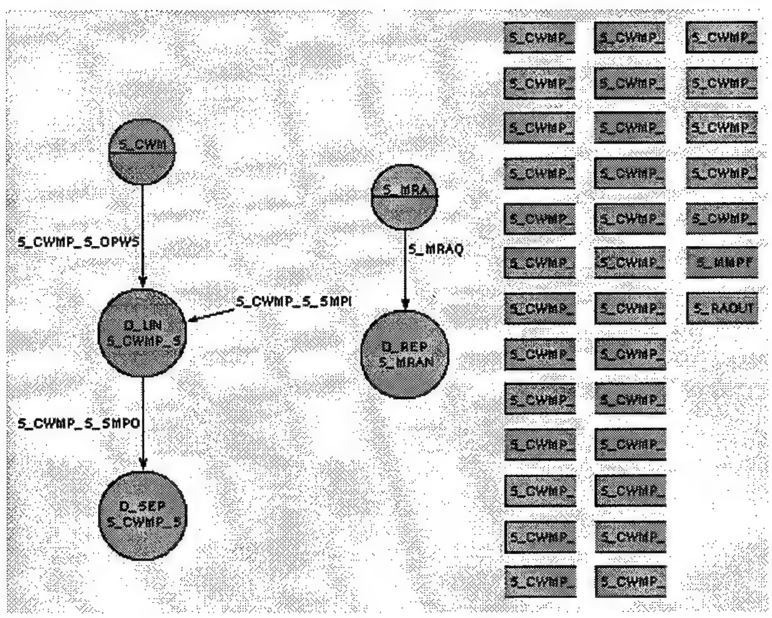


Figure 95. Partition P_CWM_1K

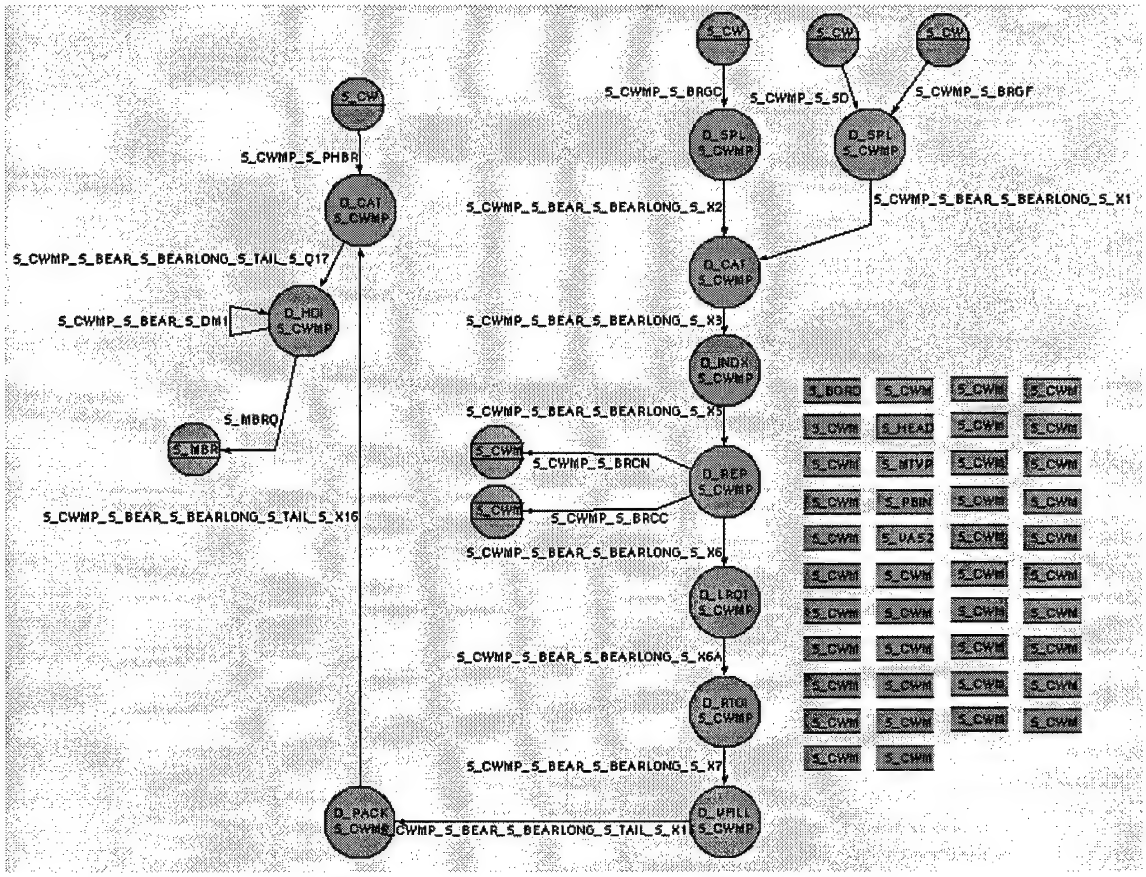


Figure 96. Partition P_CWM_1L

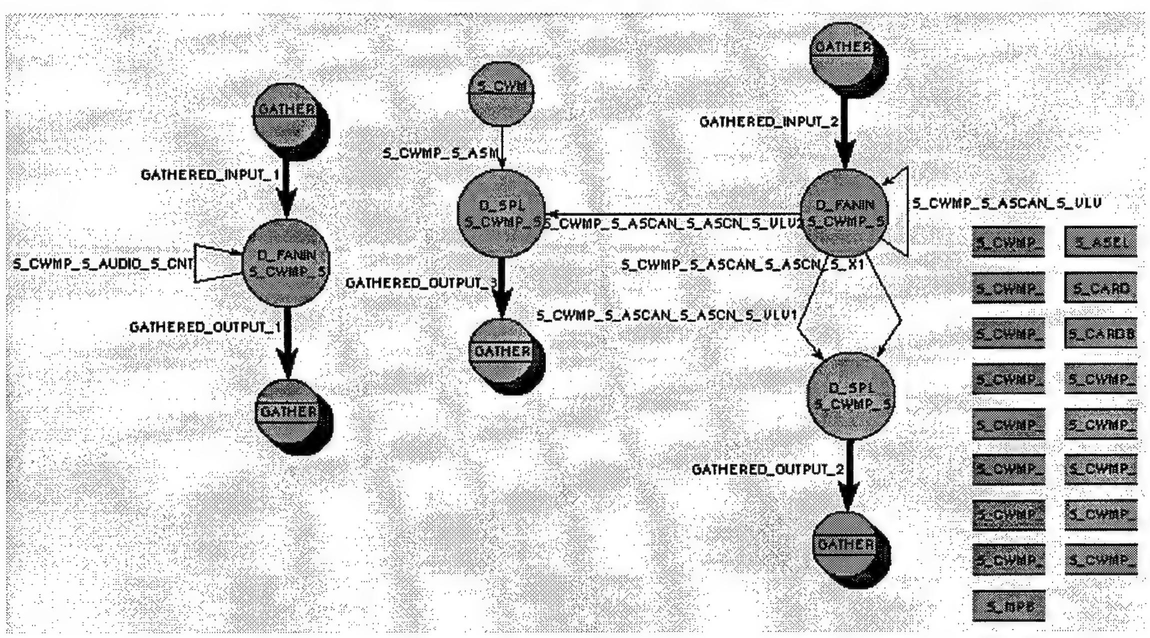


Figure 97. Partition P_CWM_2

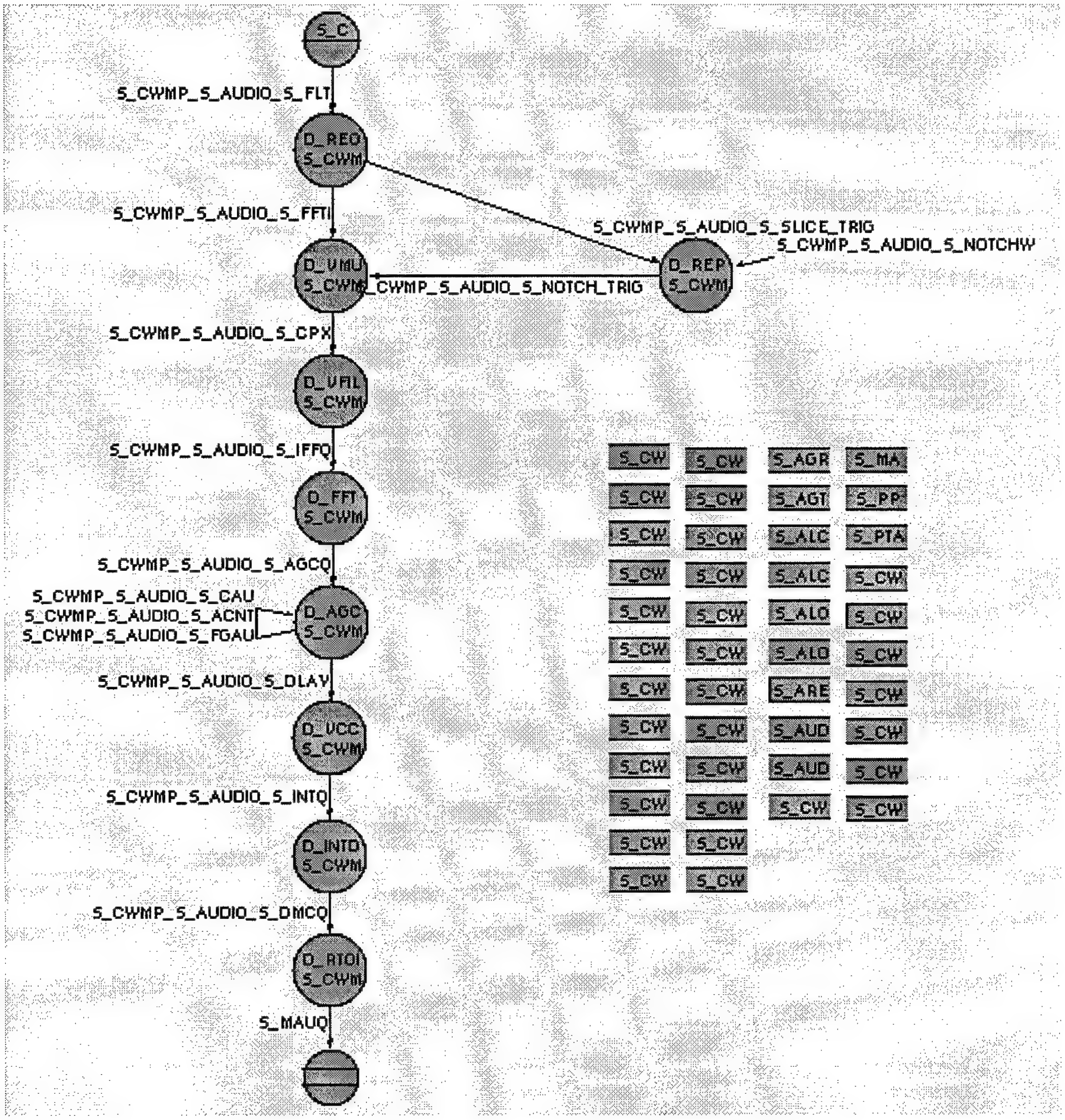


Figure 98. Partition P_CWM_2B

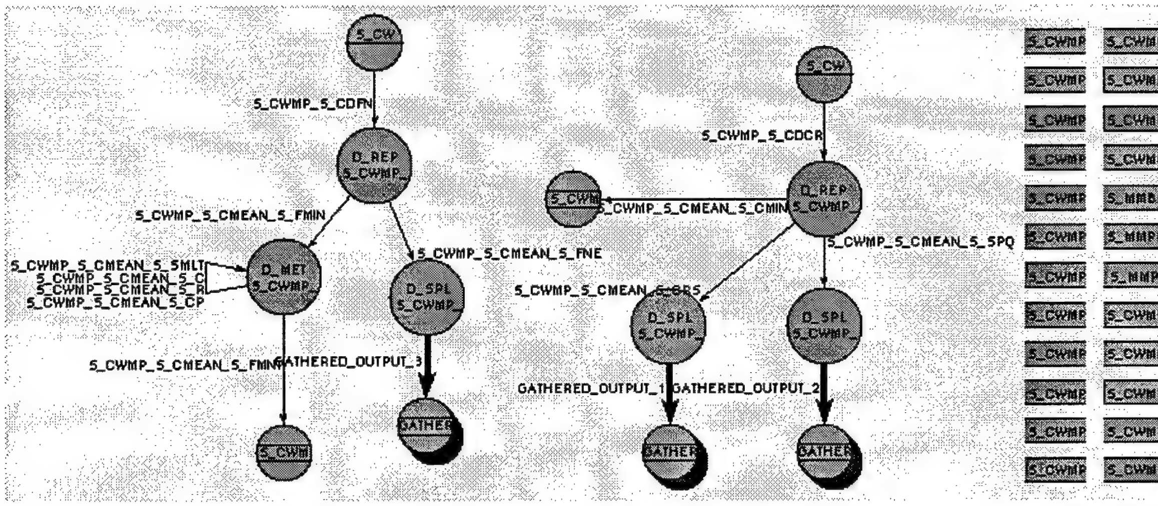


Figure 99. Partition P_CWM_3

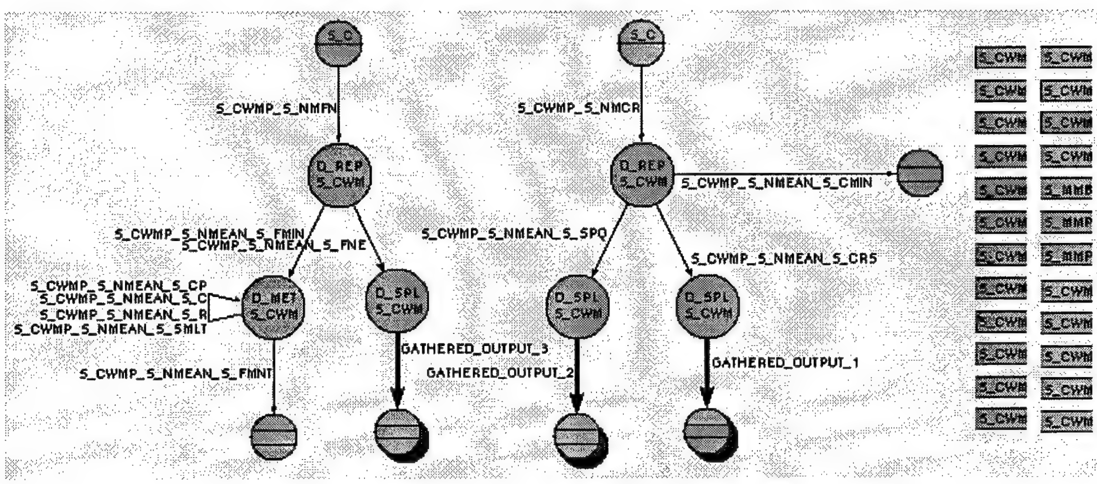


Figure 100. Partition P_CWM_3A

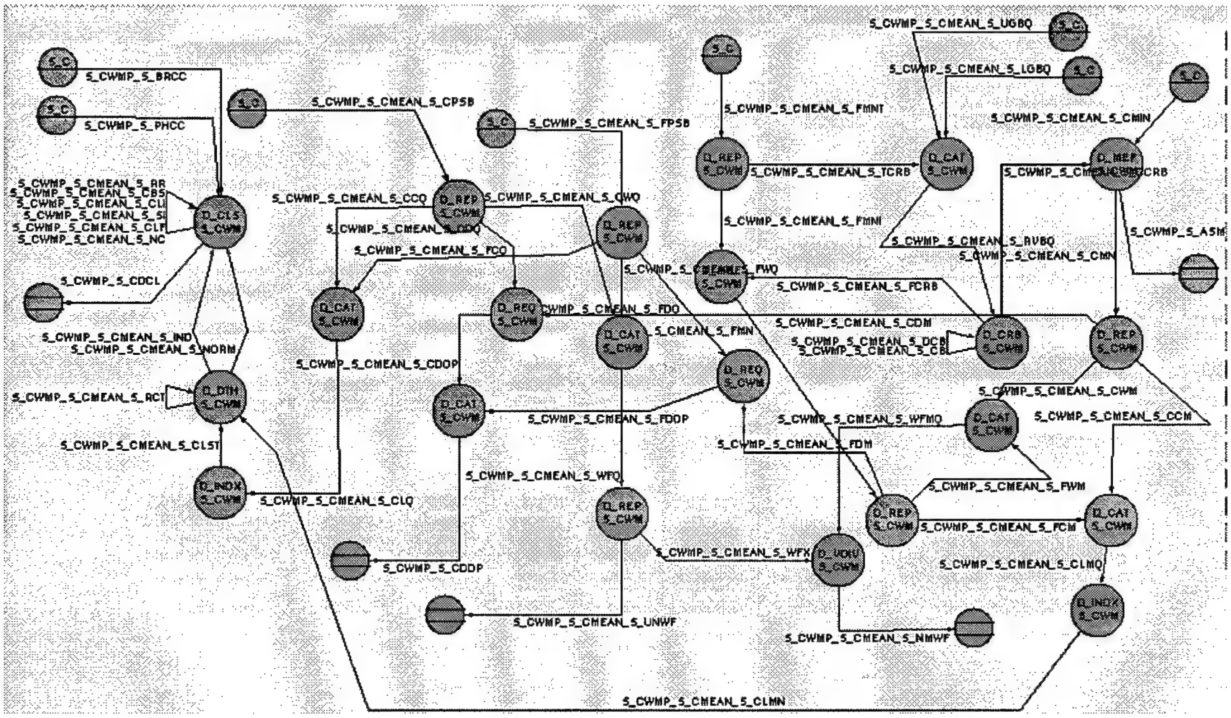


Figure 101. Partition P_CWM_3B

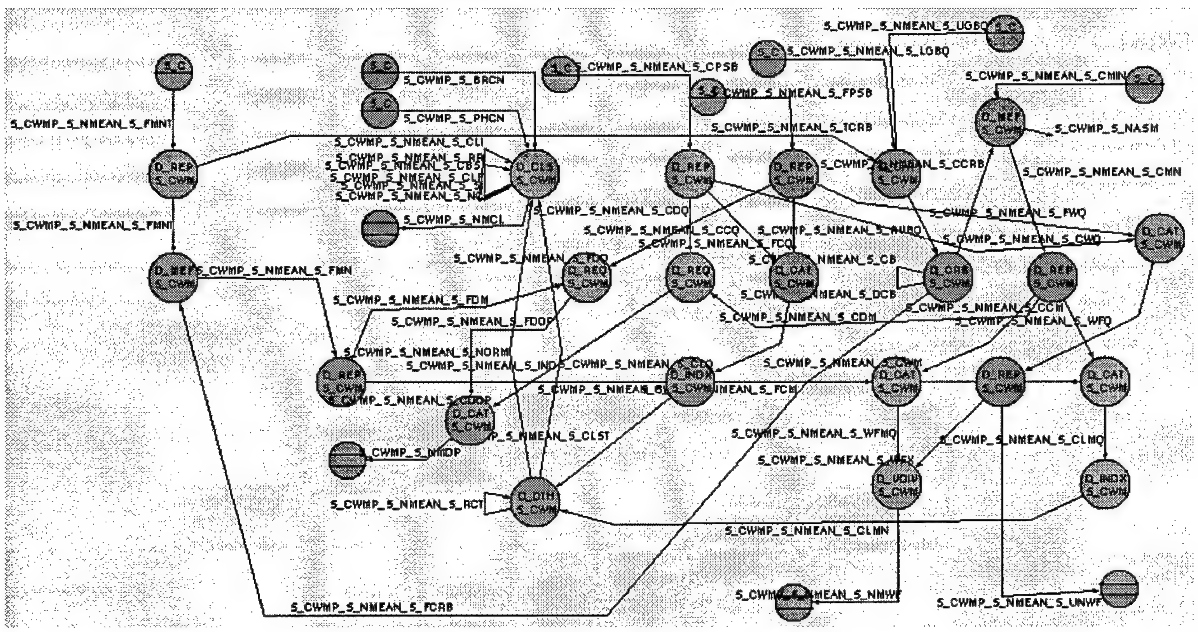


Figure 102. Partition P_CWM_3C

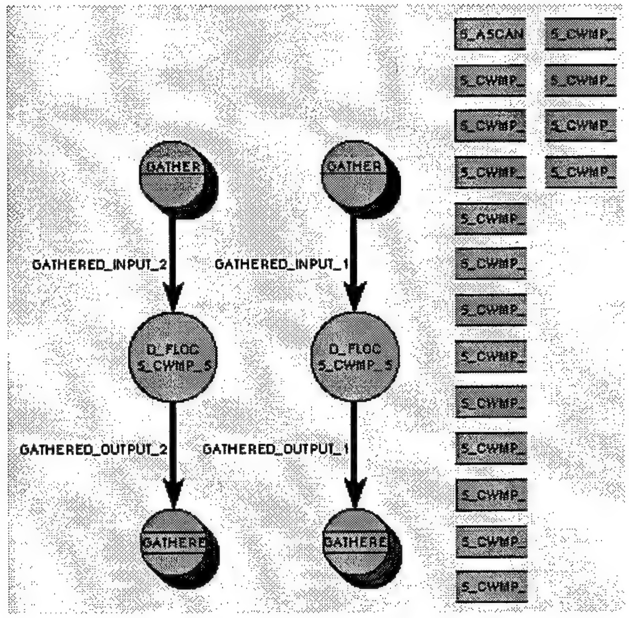


Figure 103. Partition P_CWM_3D

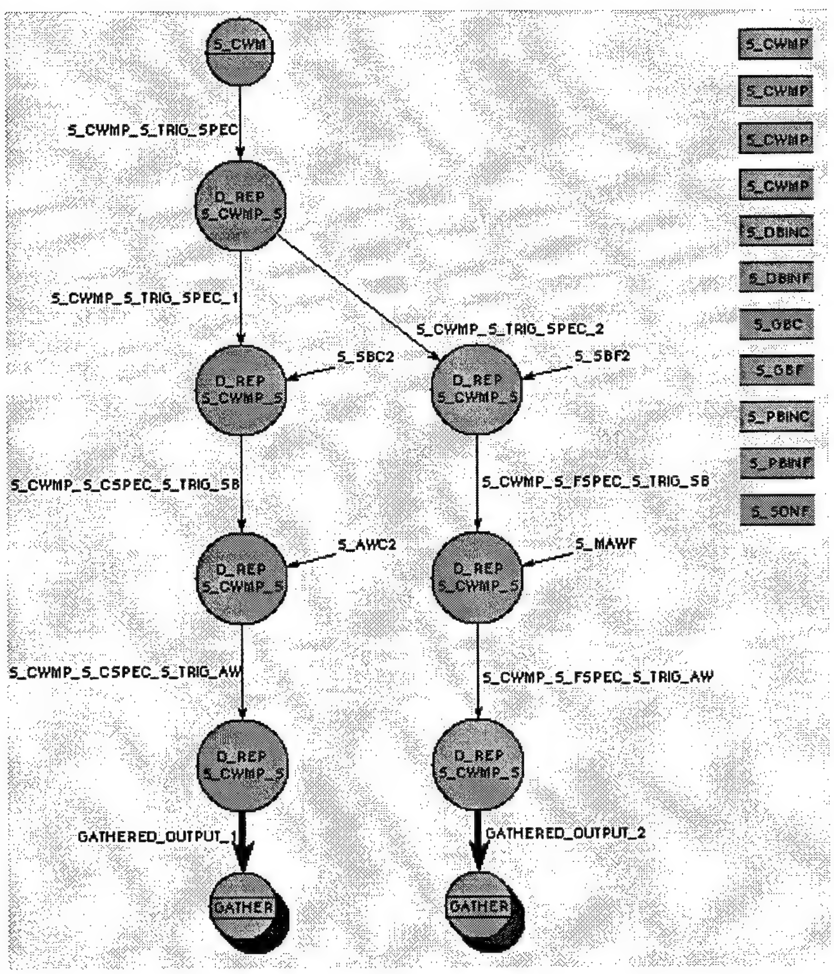


Figure 104. Partition P_CWM_4

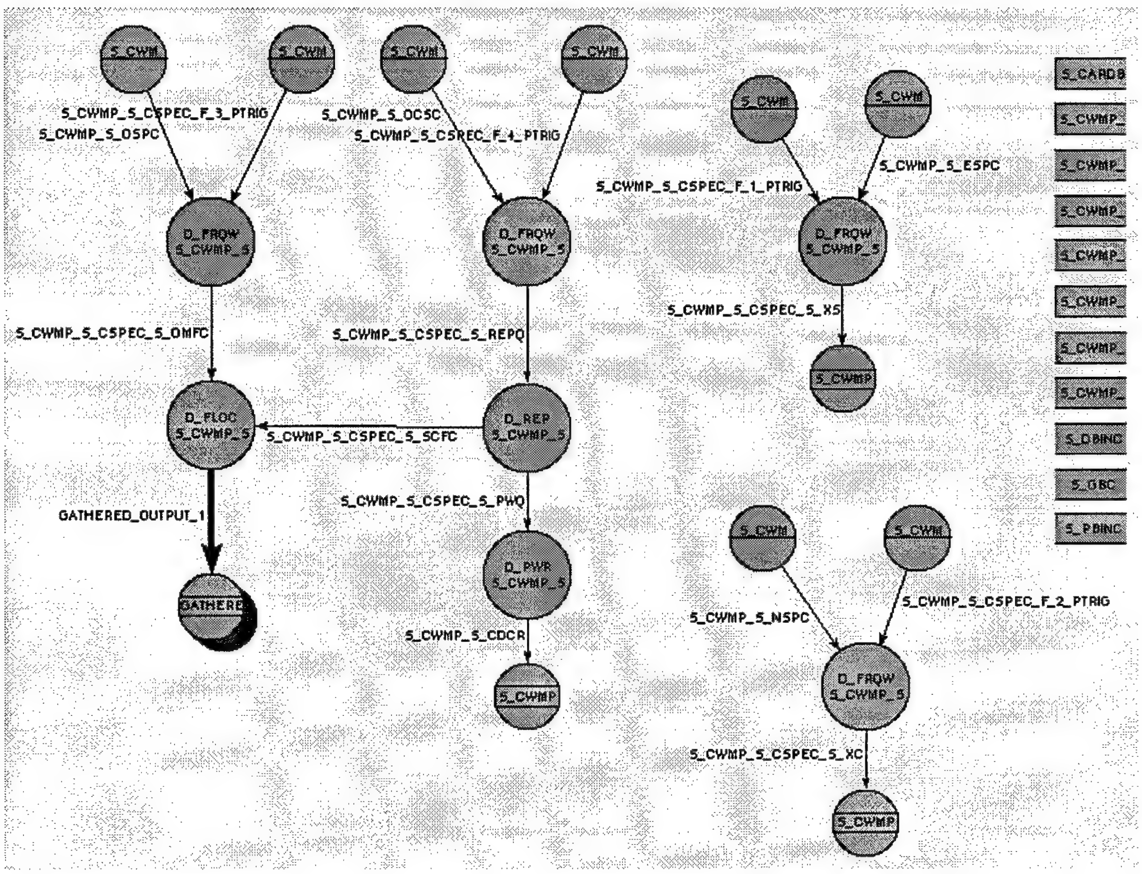


Figure 105. Partition P_CWM_4A

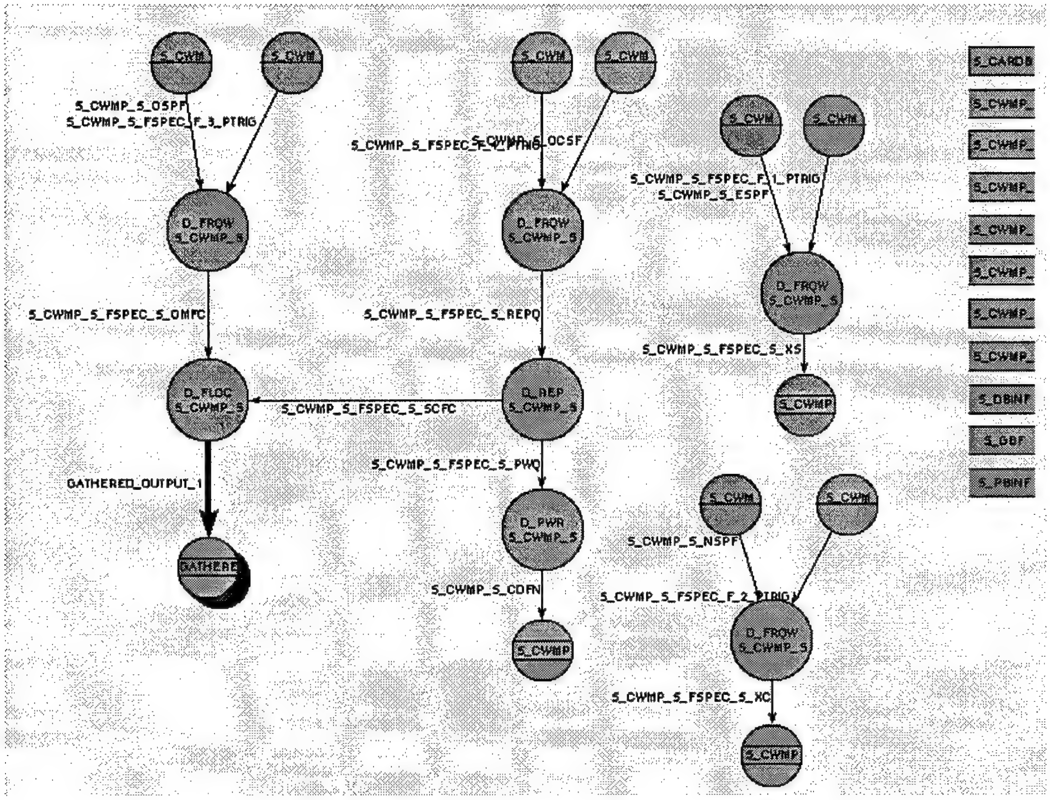


Figure 106. Partition P_CWM_4B

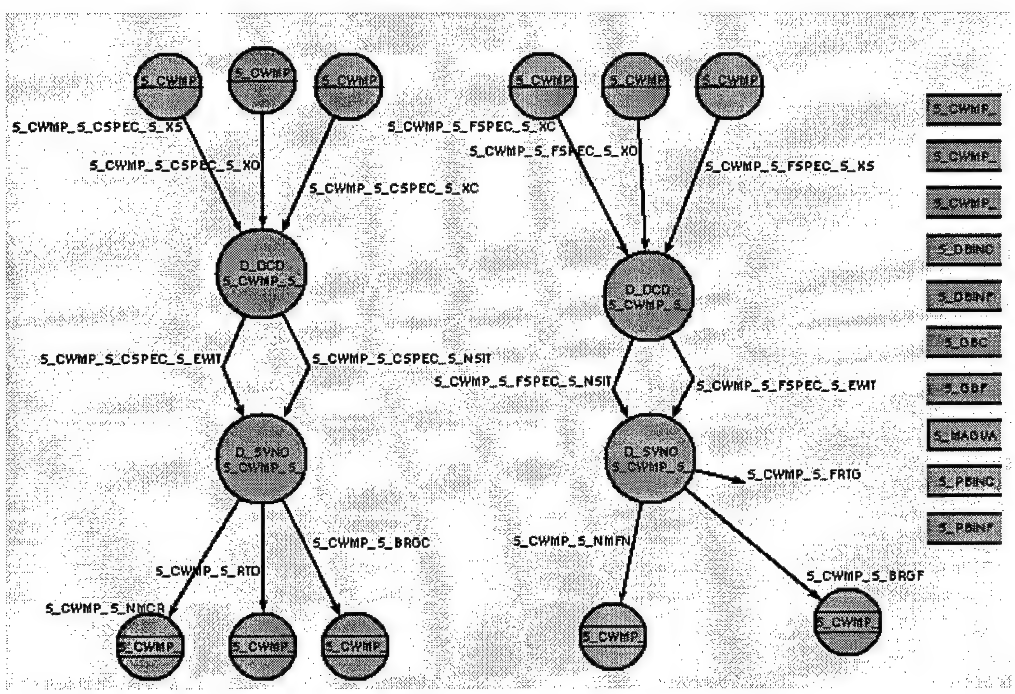


Figure 107. Partition P_CWM_4C

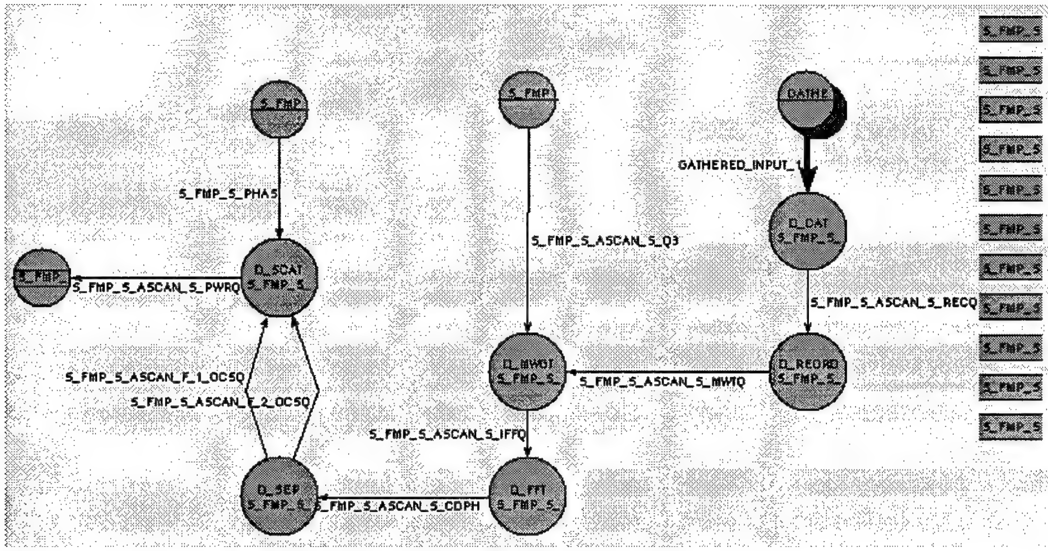


Figure 108. Partition P_FMASCAN_1

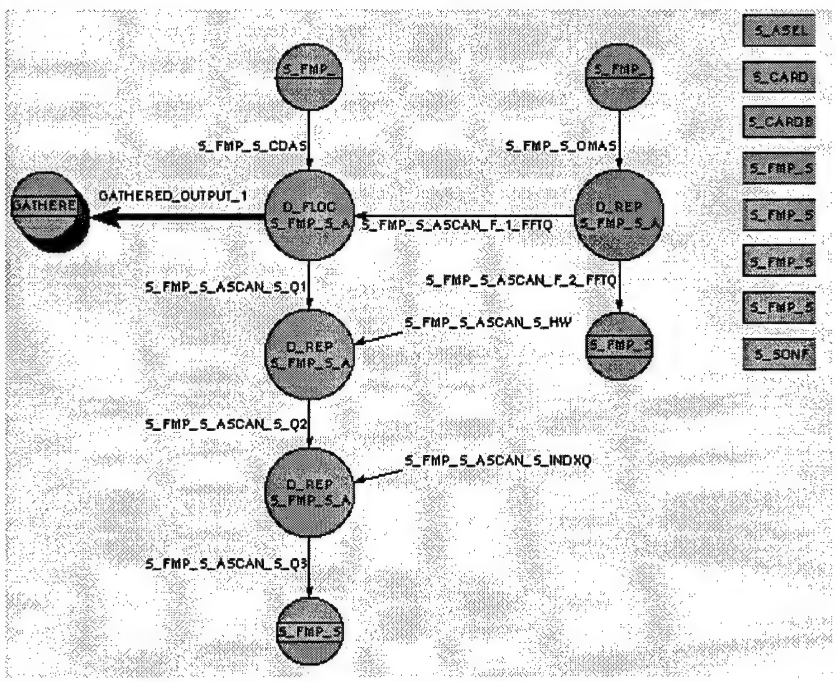


Figure 109. Partition P_FMASCAN_1A

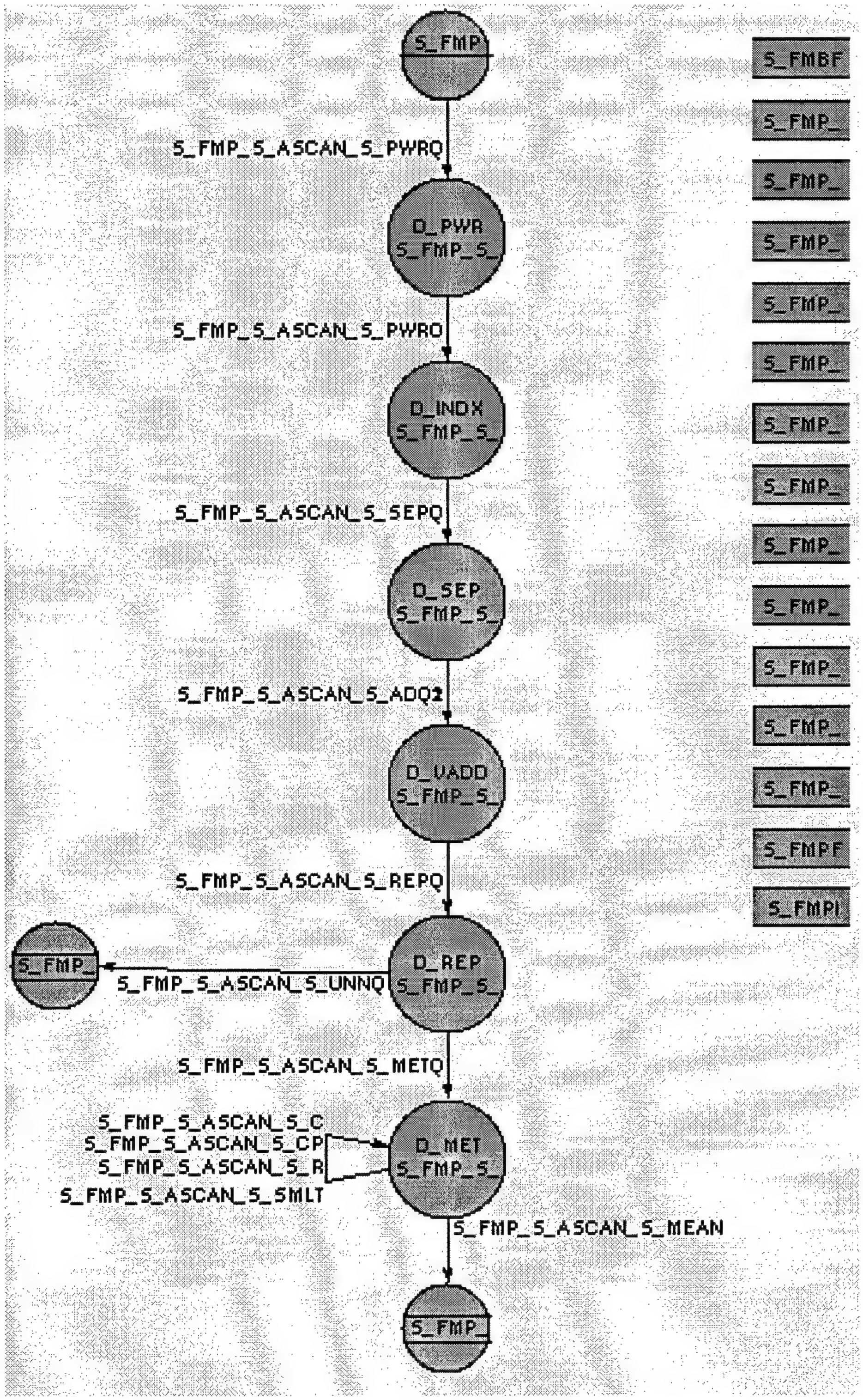


Figure 110. Partition P_FMASCAN_2

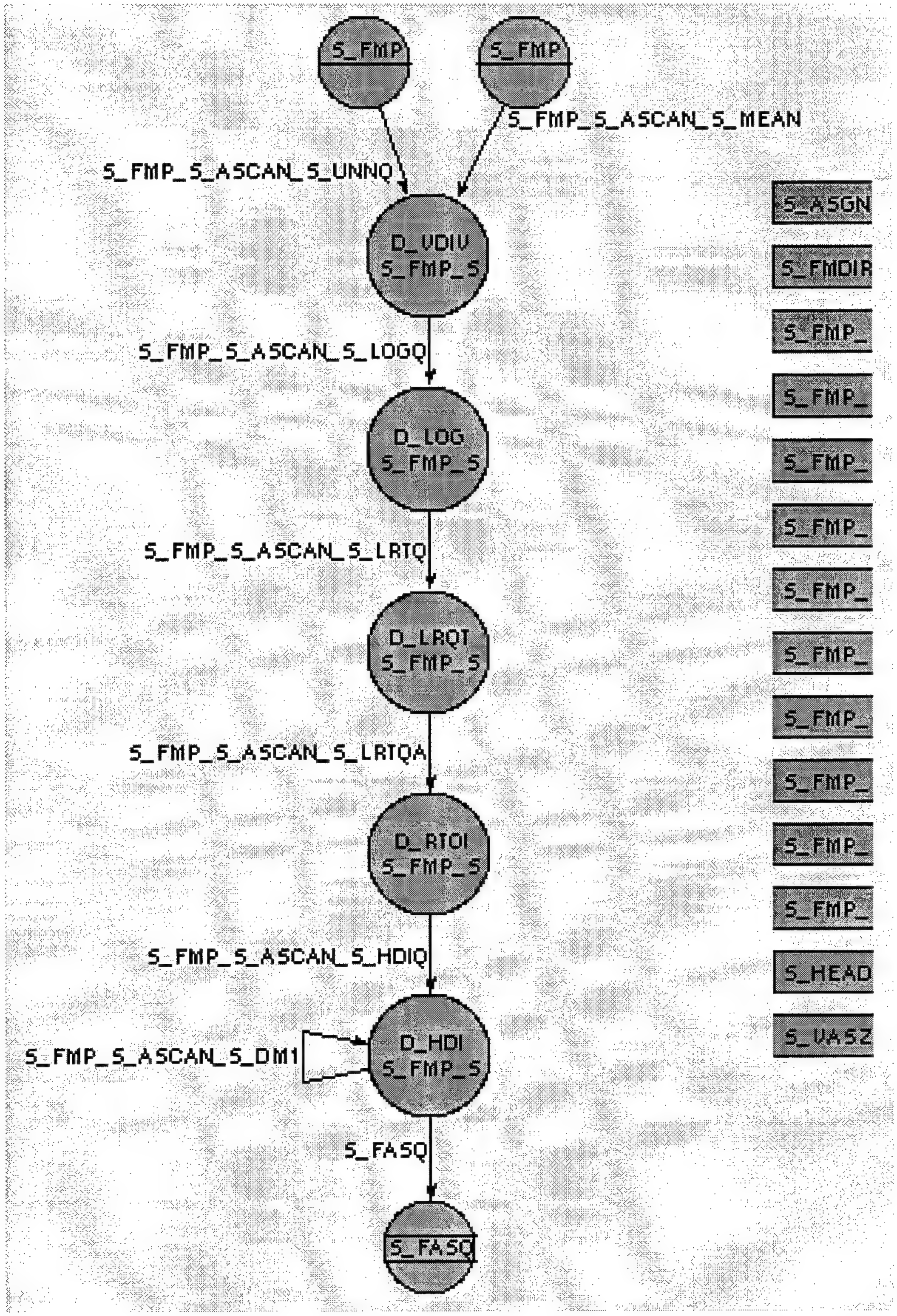


Figure 111. Partition P_FMASCAN_2A

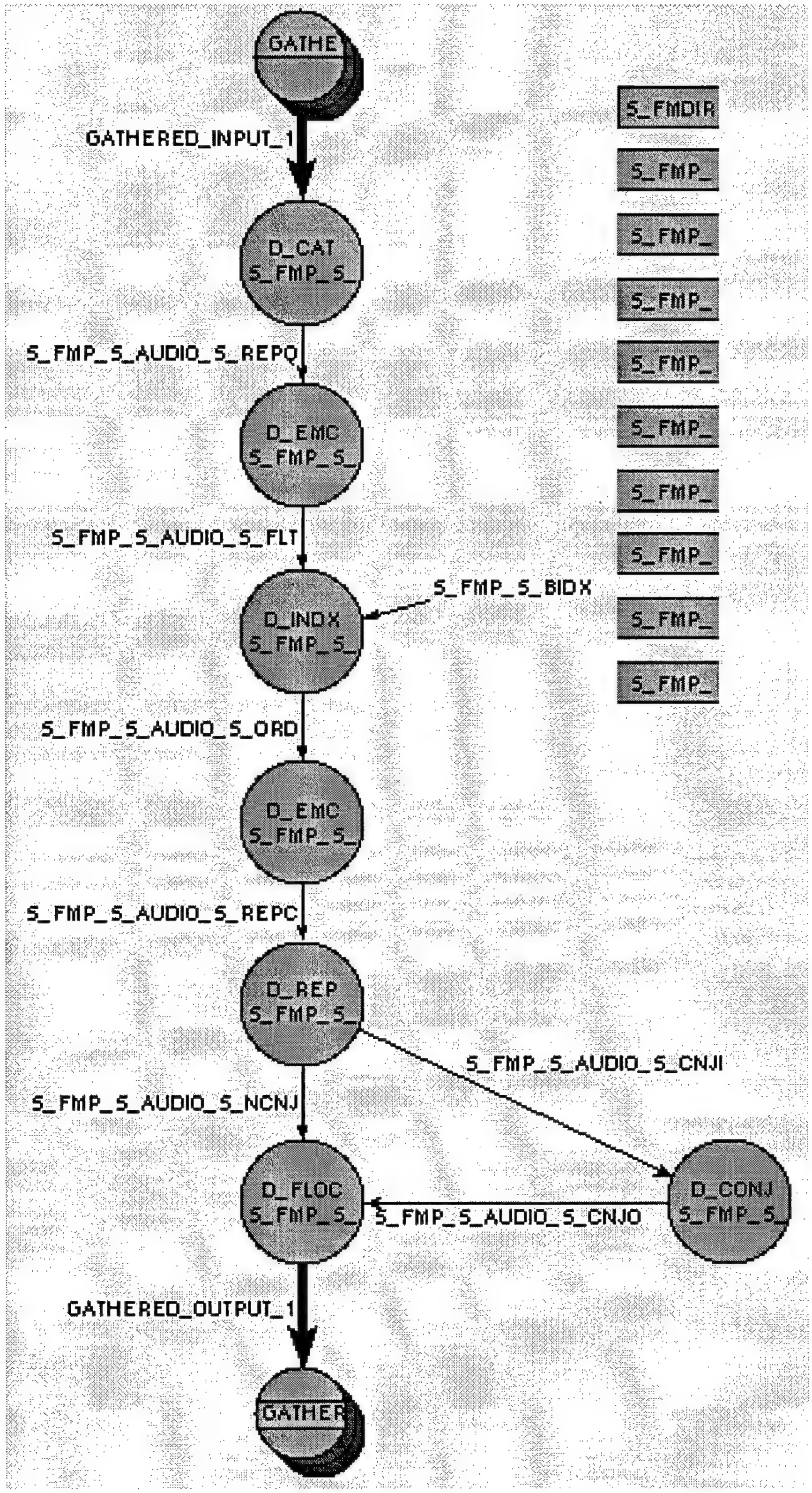


Figure 112. Partition P_FMAUD_1

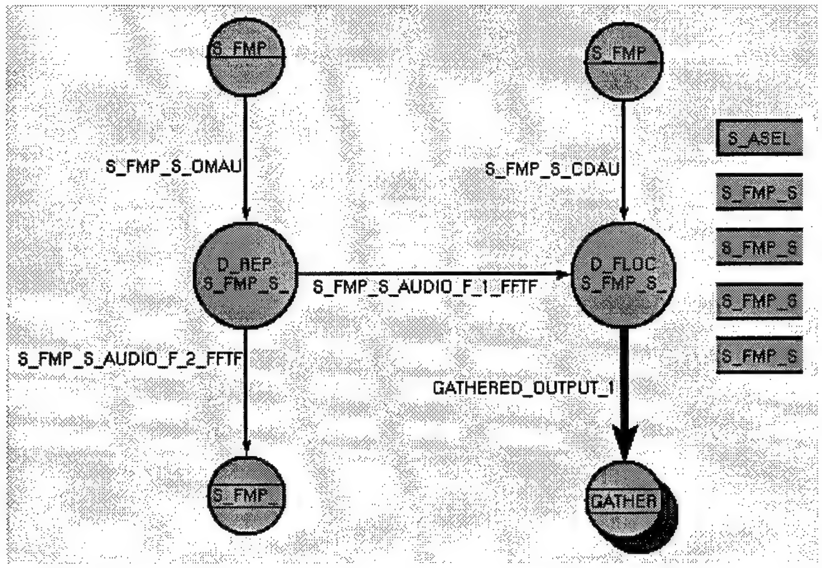


Figure 113. Partition P_FMAUD_1A

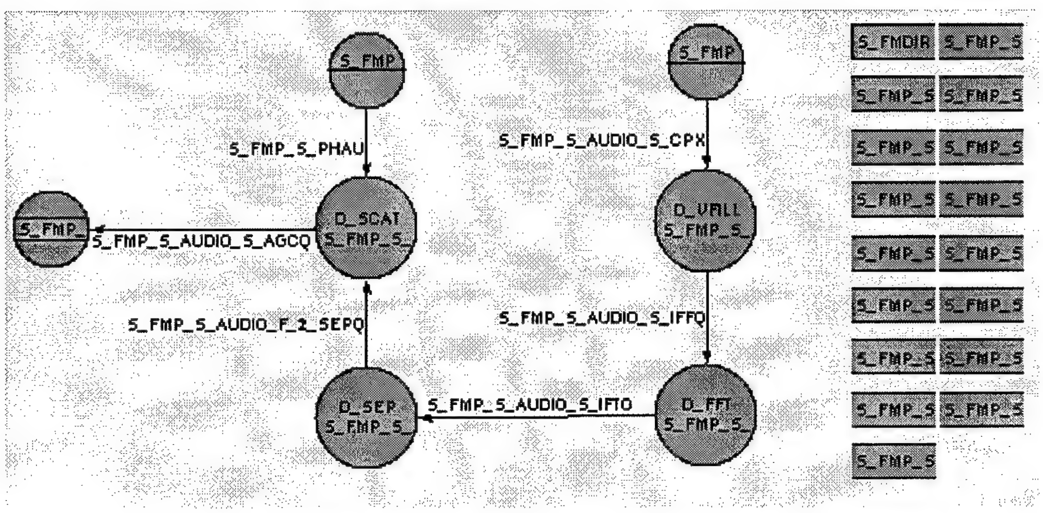


Figure 114. Partition P_FMAUD_1B

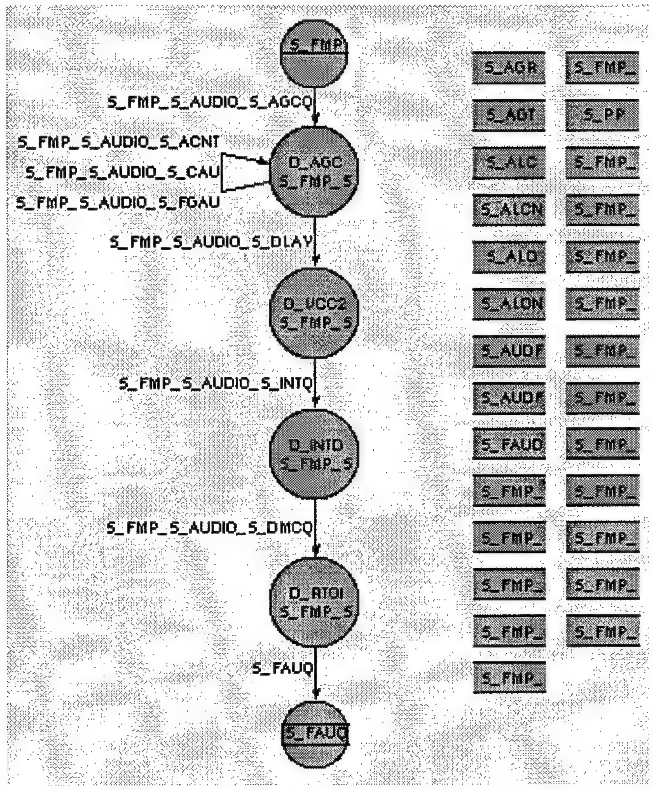


Figure 115. Partition P_FMAUD_2

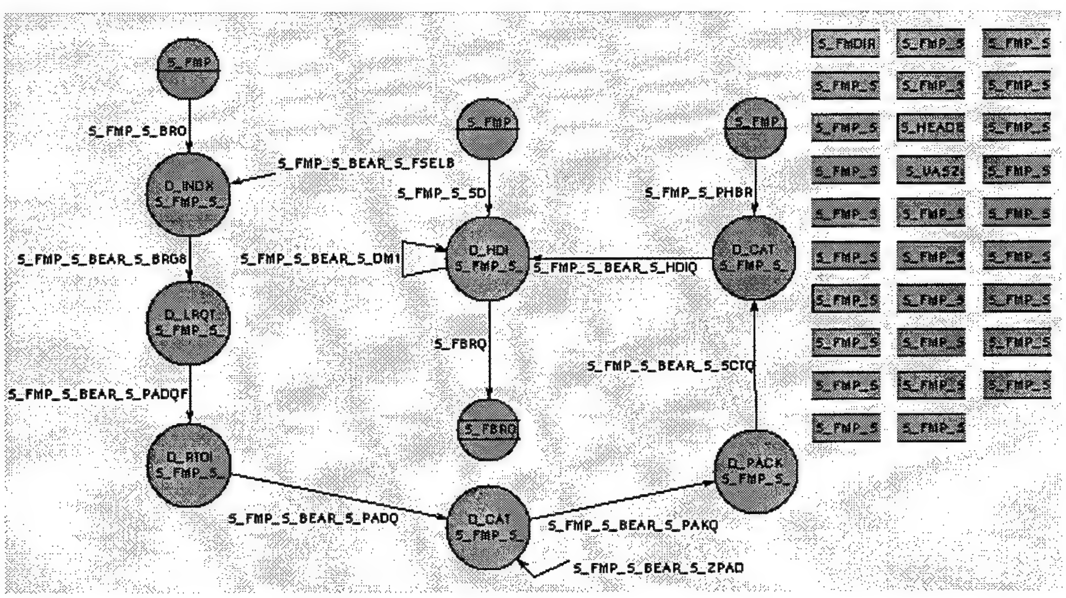


Figure 116. Partition P_FMBR

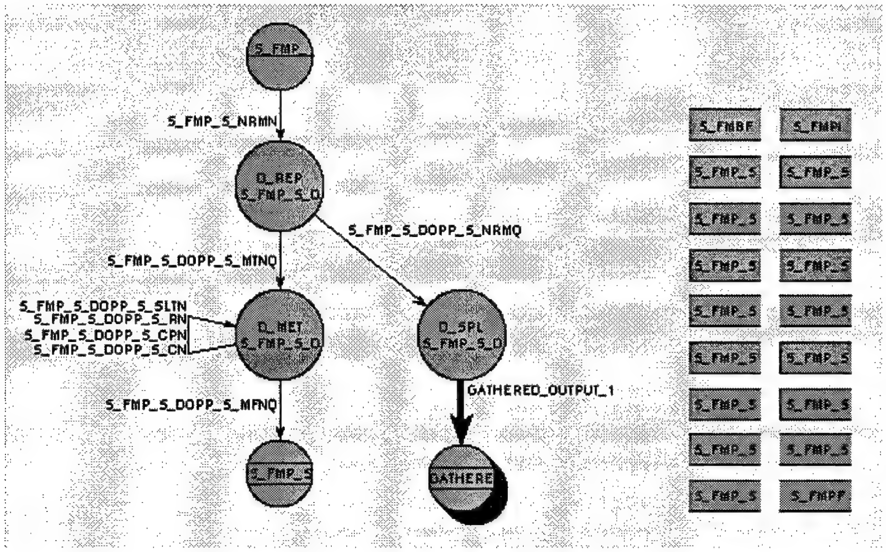


Figure 117. Partition P_FMDOP_1

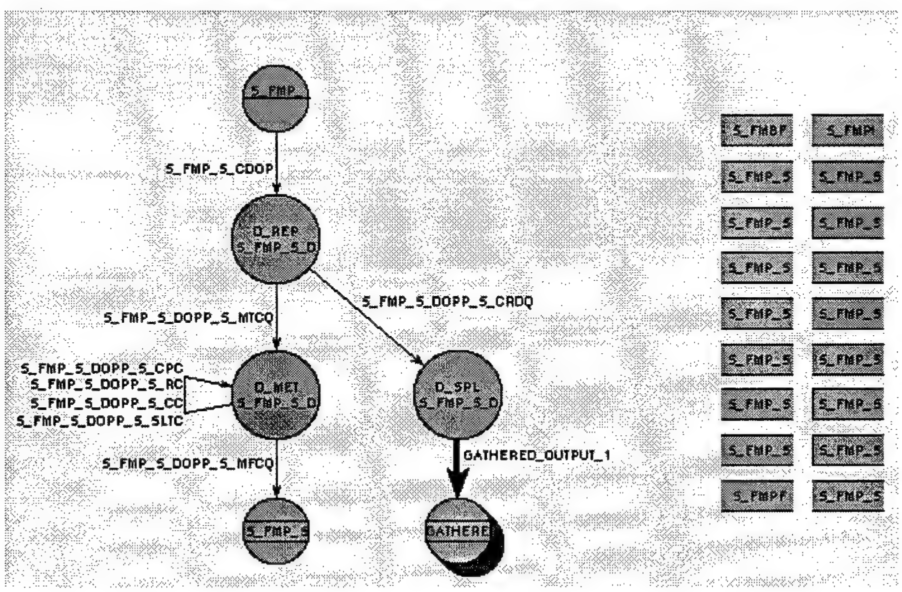


Figure 118. Partition P_FMDOP_1A

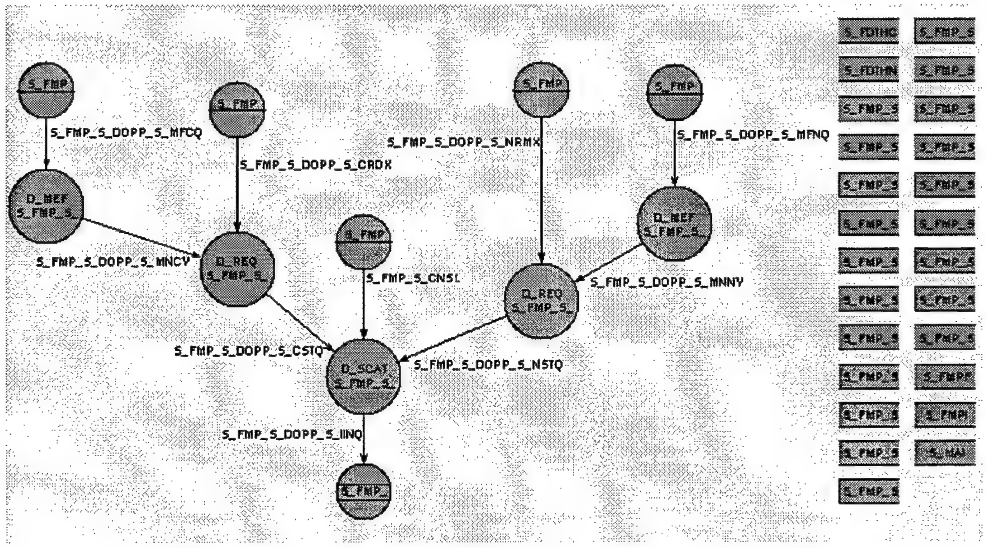


Figure 119. Partition P_FMDOP_1B

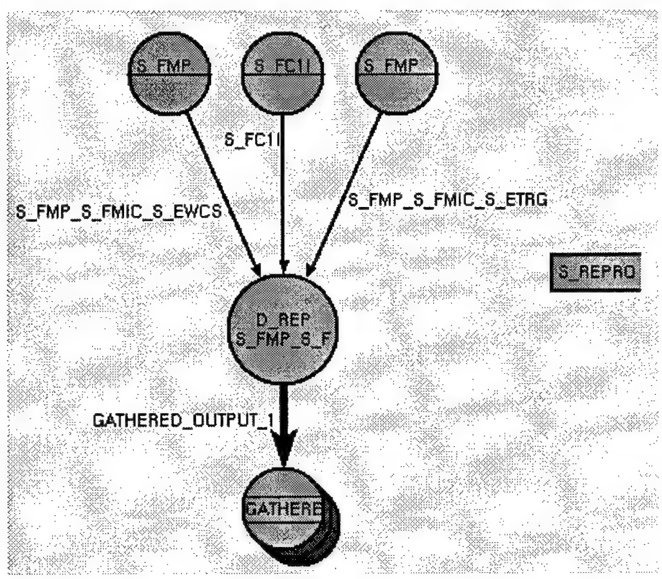


Figure 120. Partition P_FMDOP_2

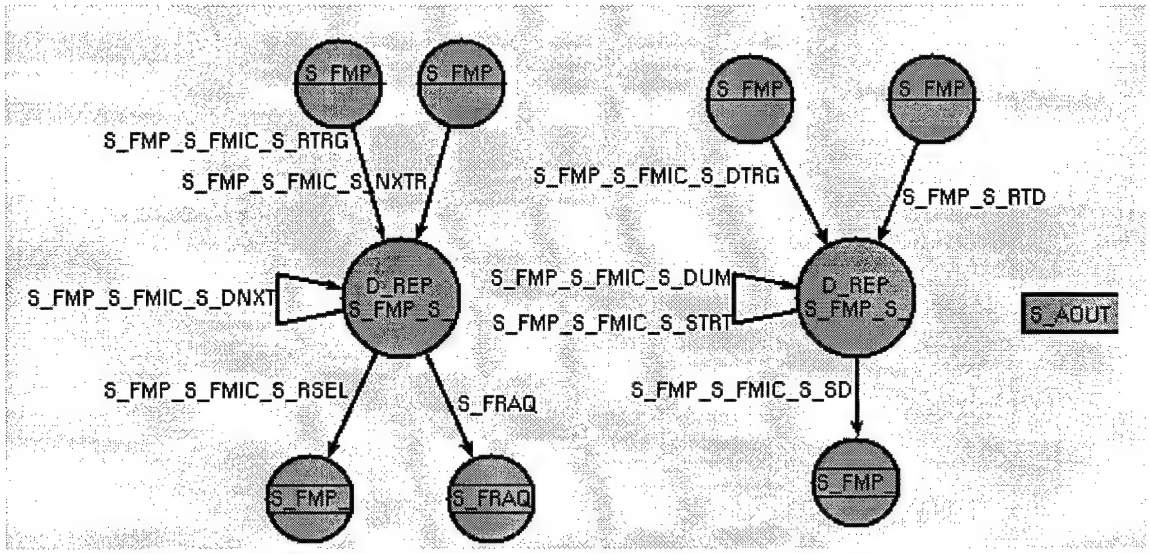


Figure 121. Partition P_FMIN_1

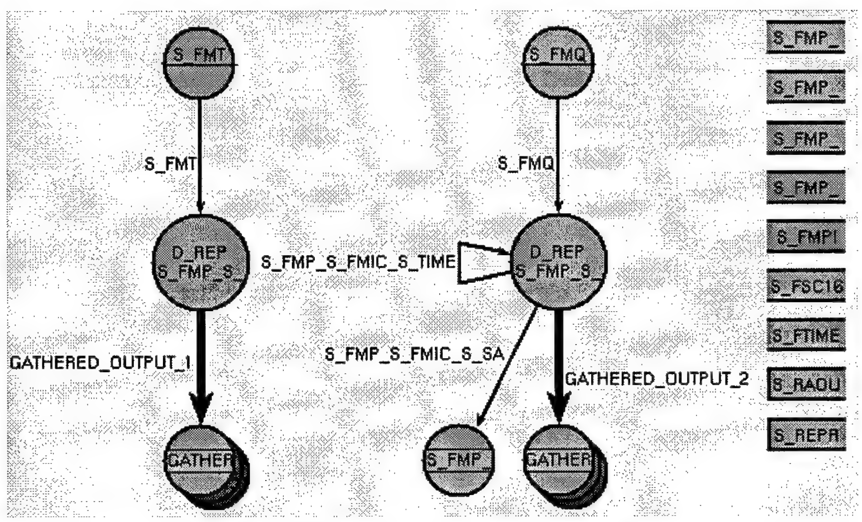


Figure 122. Partition P_FMIN_1D

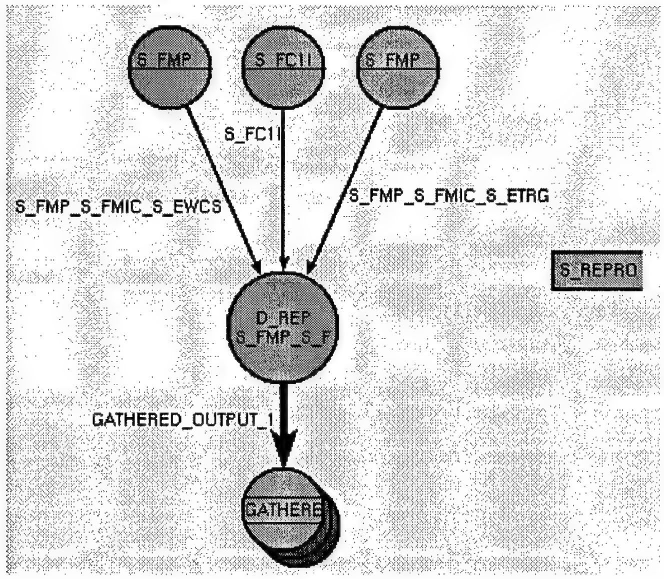


Figure 123. Partition P_FMIN_2A

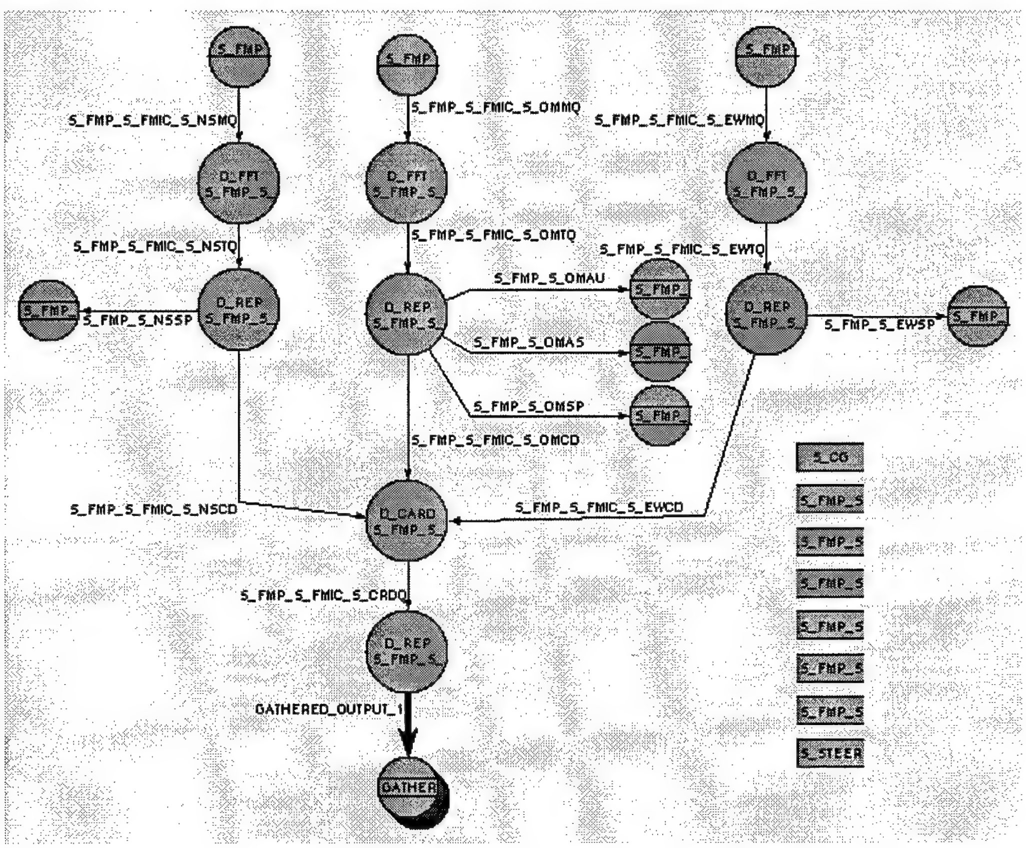


Figure 124. Partition P_FMIN_3

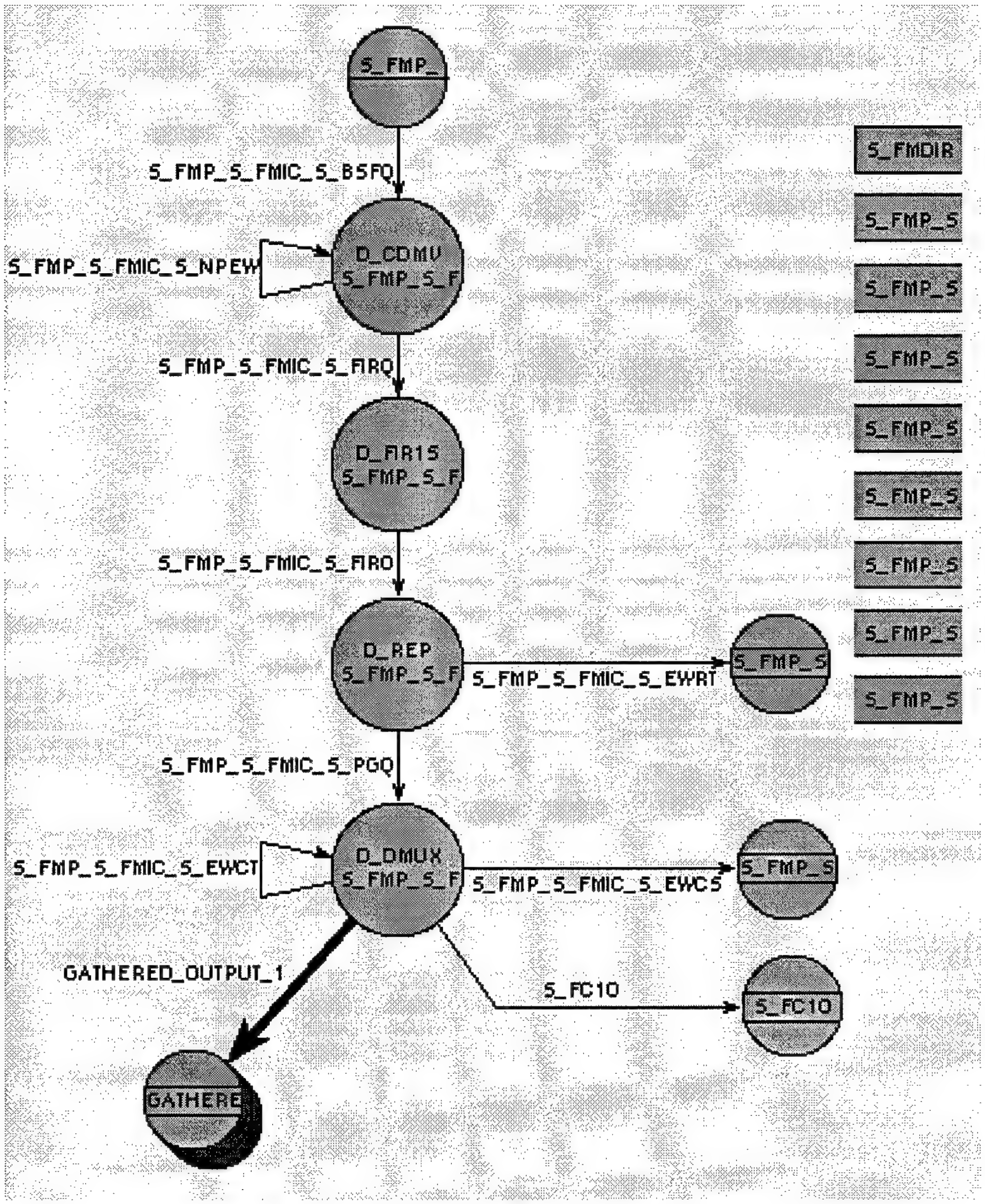


Figure 125. Partition P_FMIN_3A

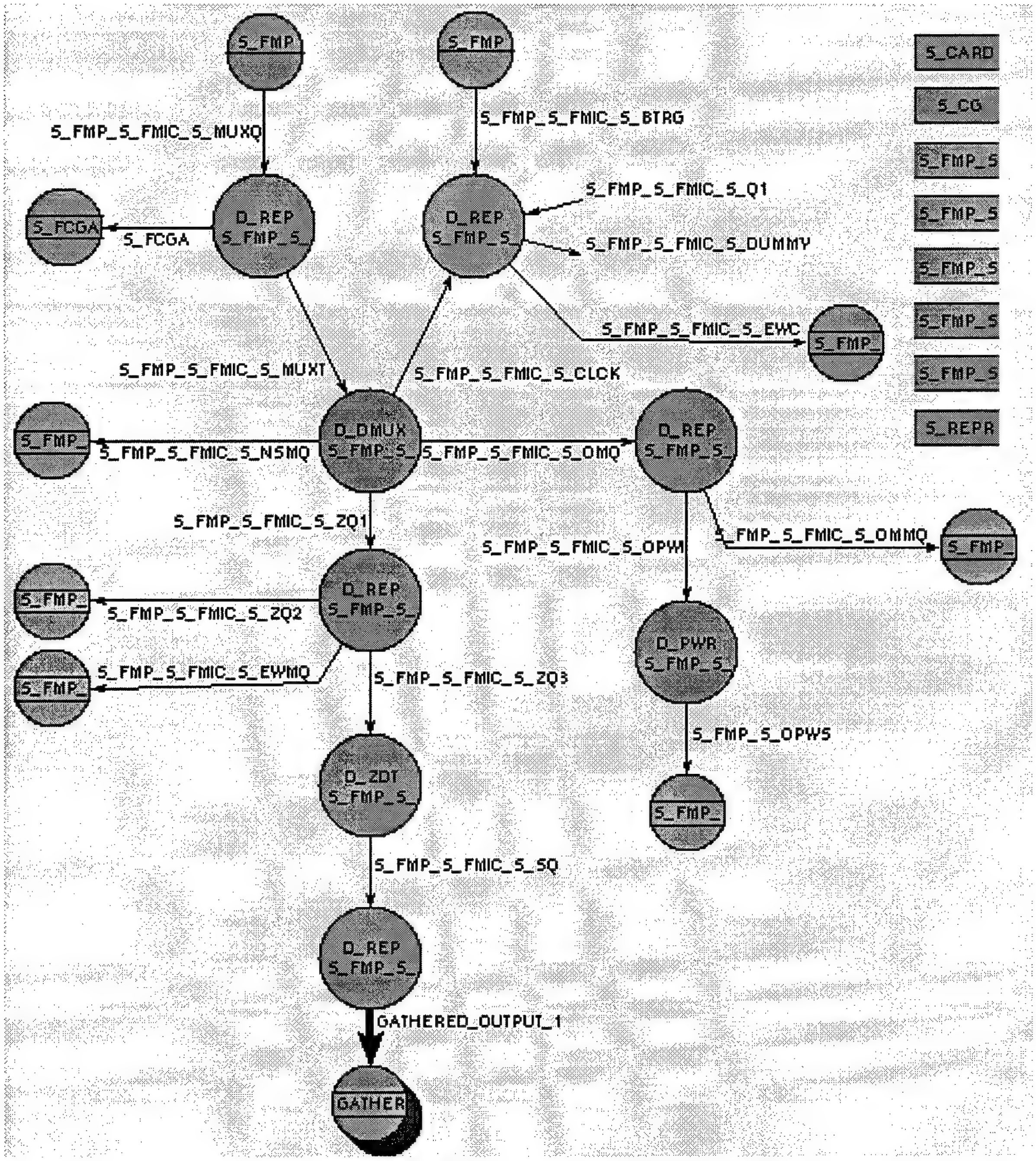


Figure 126. Partition P_FMIN_3B

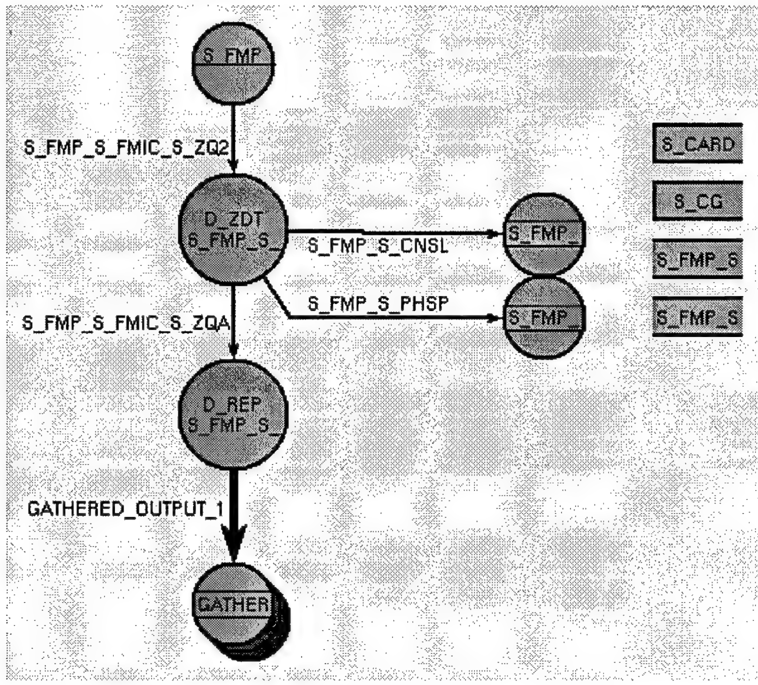


Figure 127. Partition P_FMIN_3C

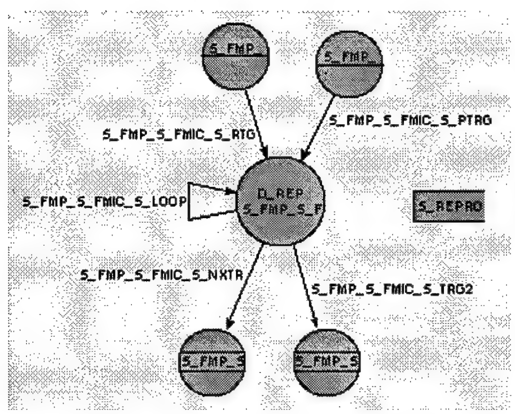


Figure 128. Partition P_FMIN_4

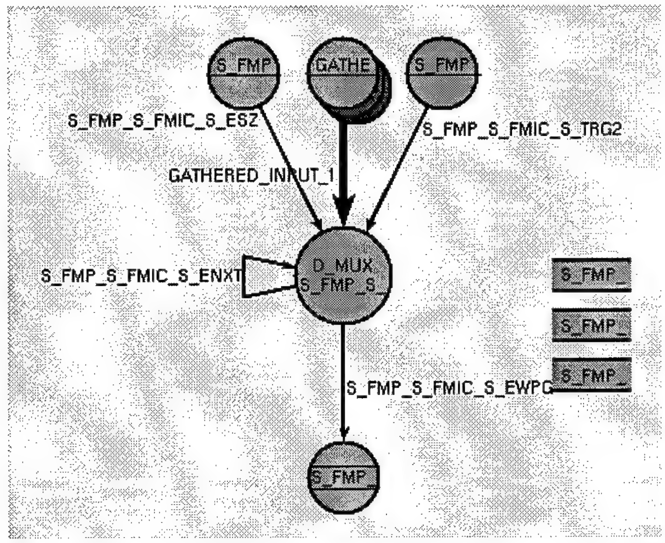


Figure 129. Partition P_FMIN_4A

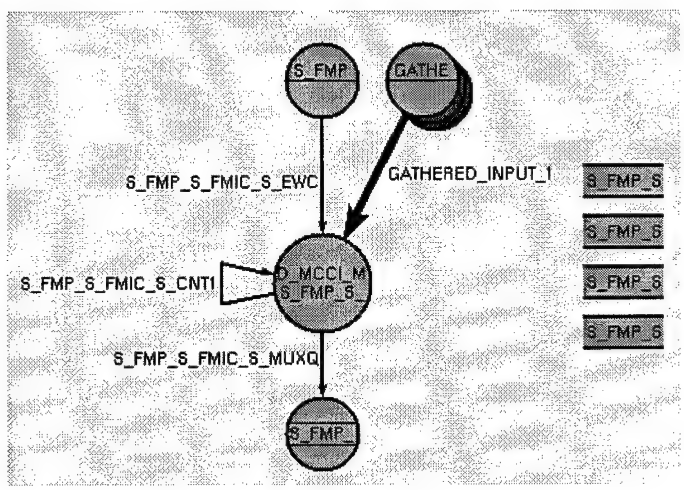


Figure 130. Partition P_FMIN_5

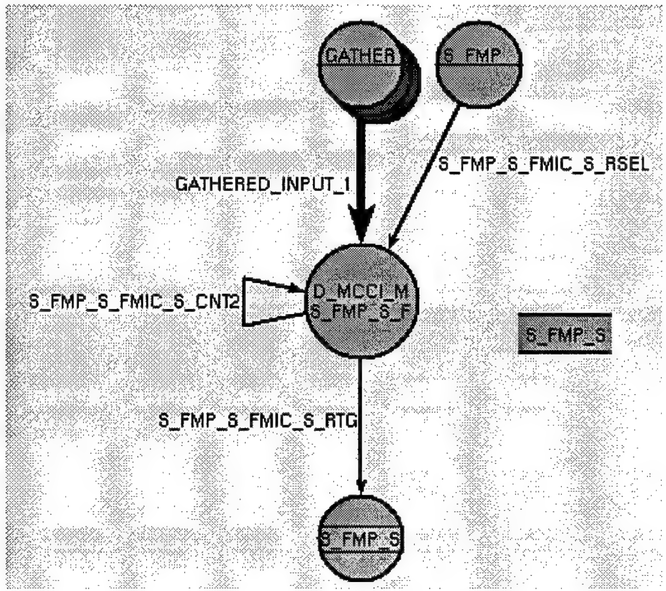


Figure 131. Partition P_FMIN_6

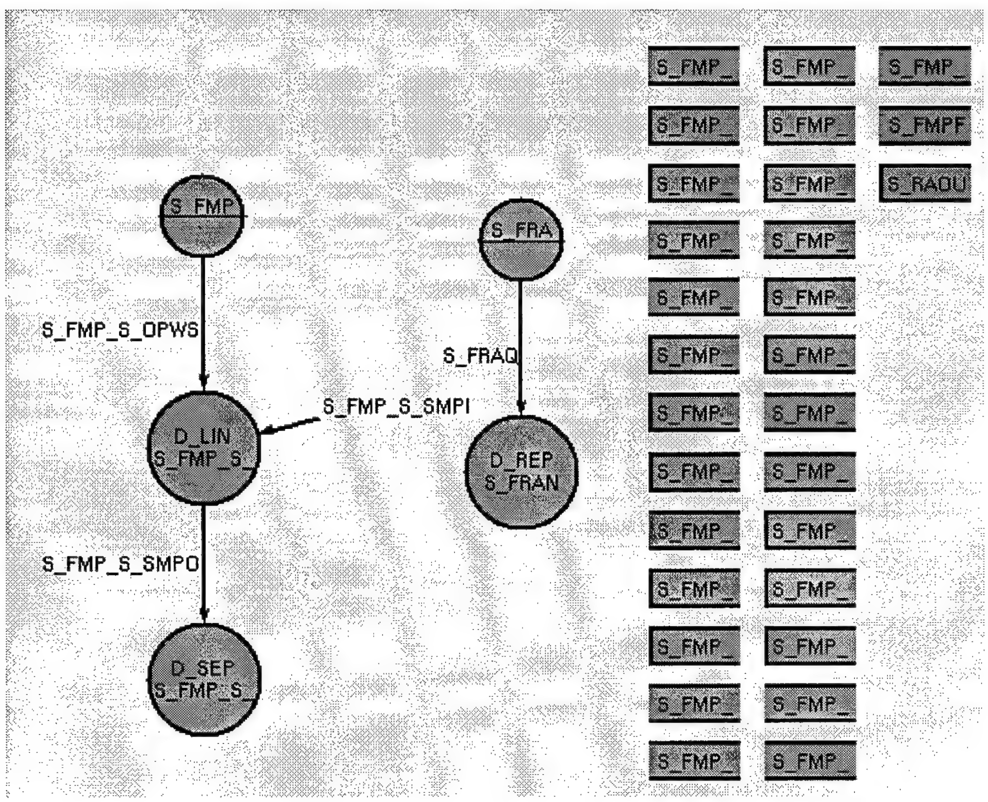


Figure 132. Partition P_FMIN_8

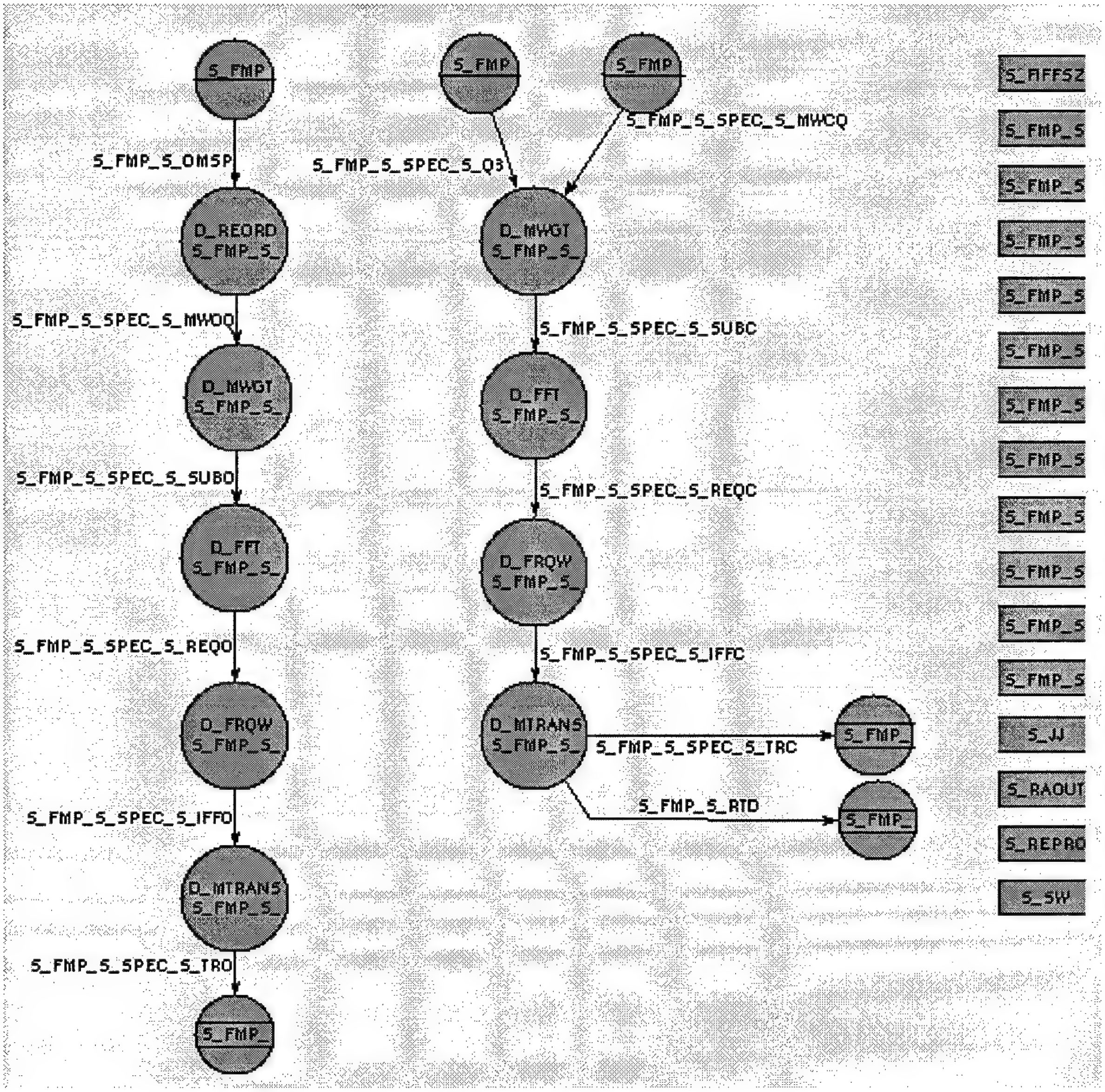


Figure 133. Partition P_FMSPEC_1

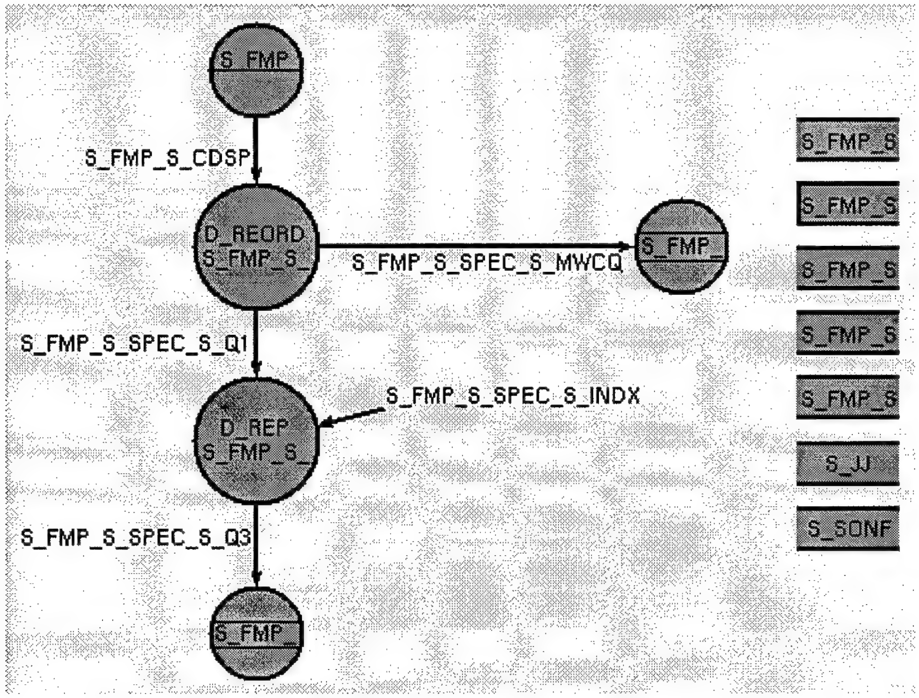


Figure 134. Partition P_FMSPEC_1A

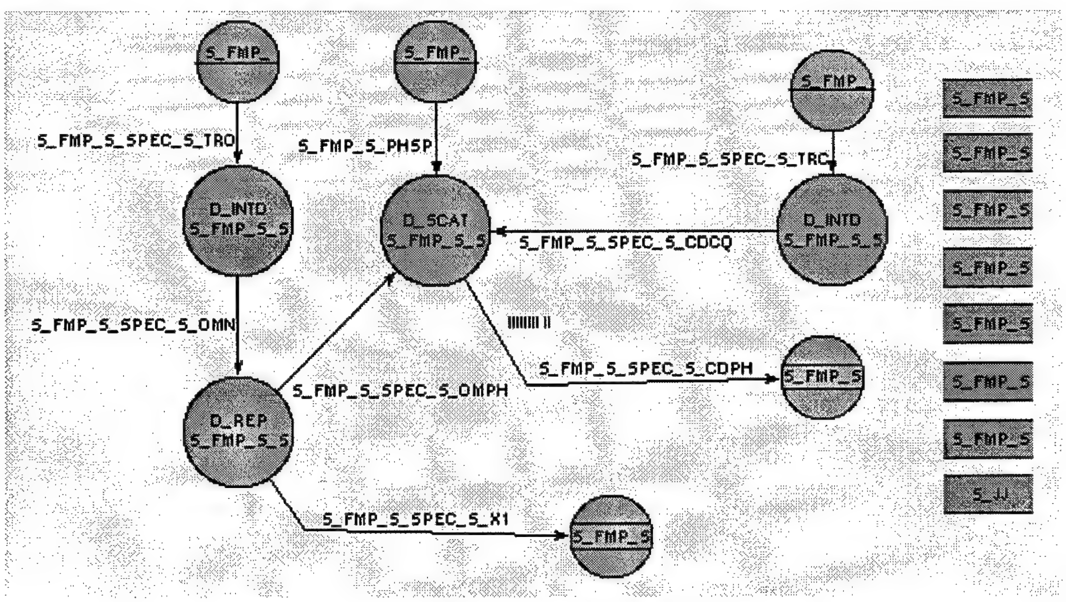


Figure 135. Partition P_FMSPEC_1B

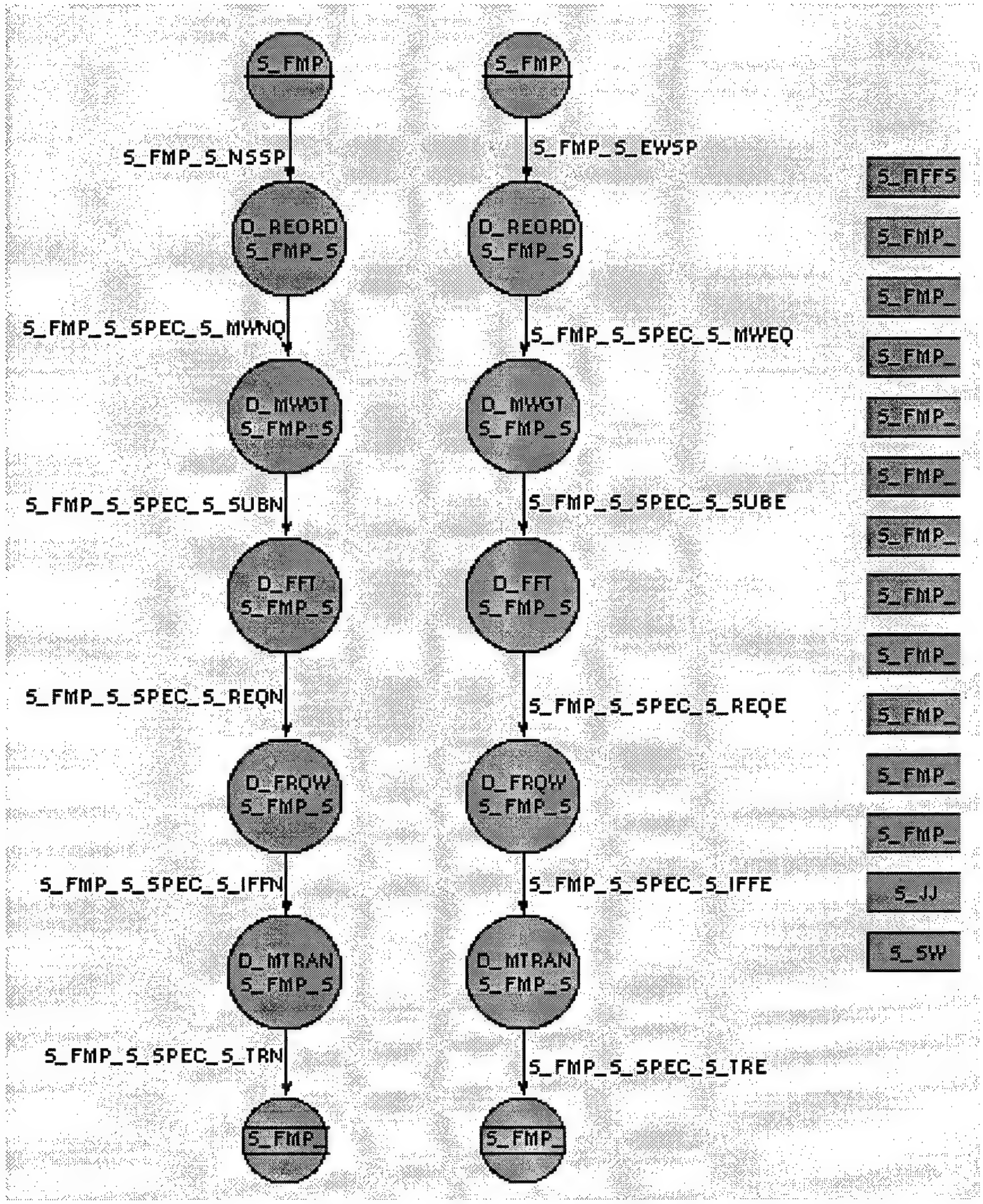


Figure 136. Partition P_FMSPEC_2

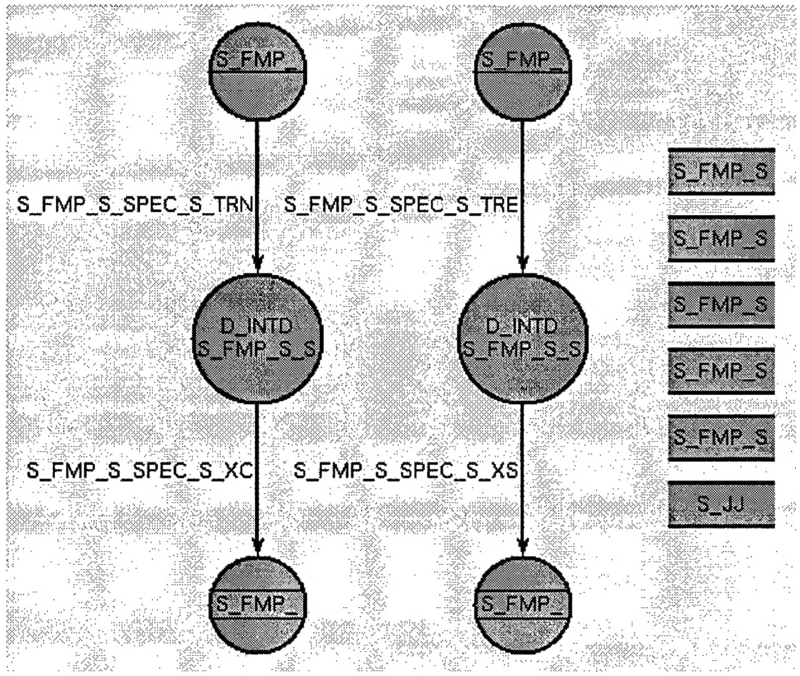


Figure 137. Partition P_FMSPEC_2A

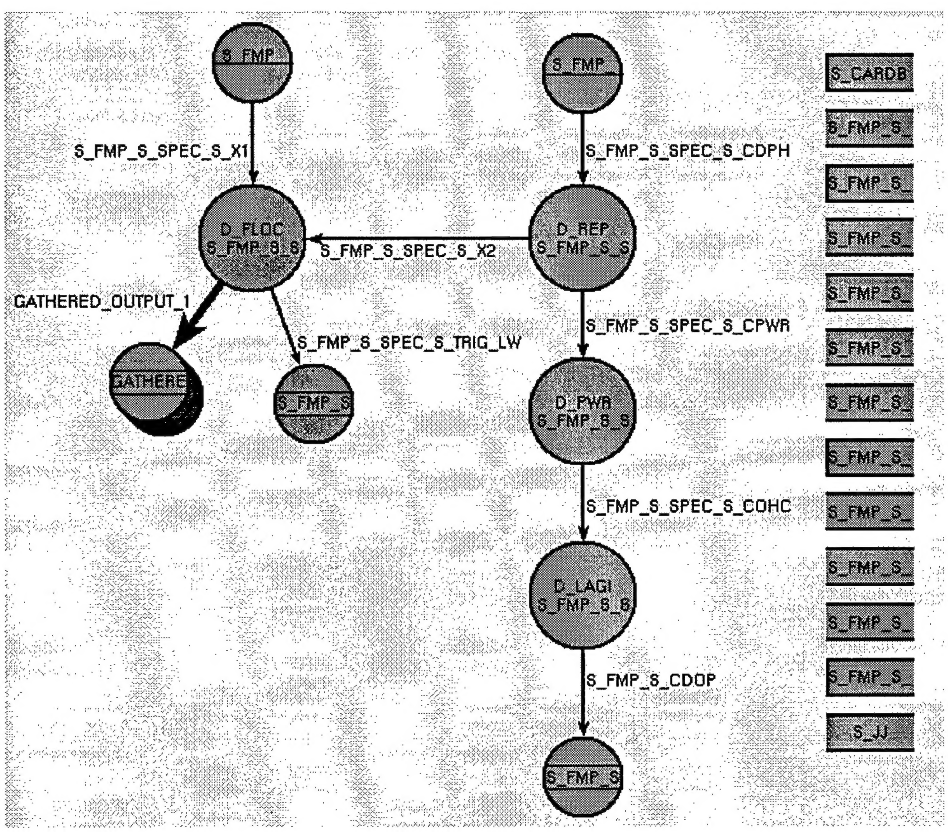


Figure 138. Partition P_FMSPEC_3

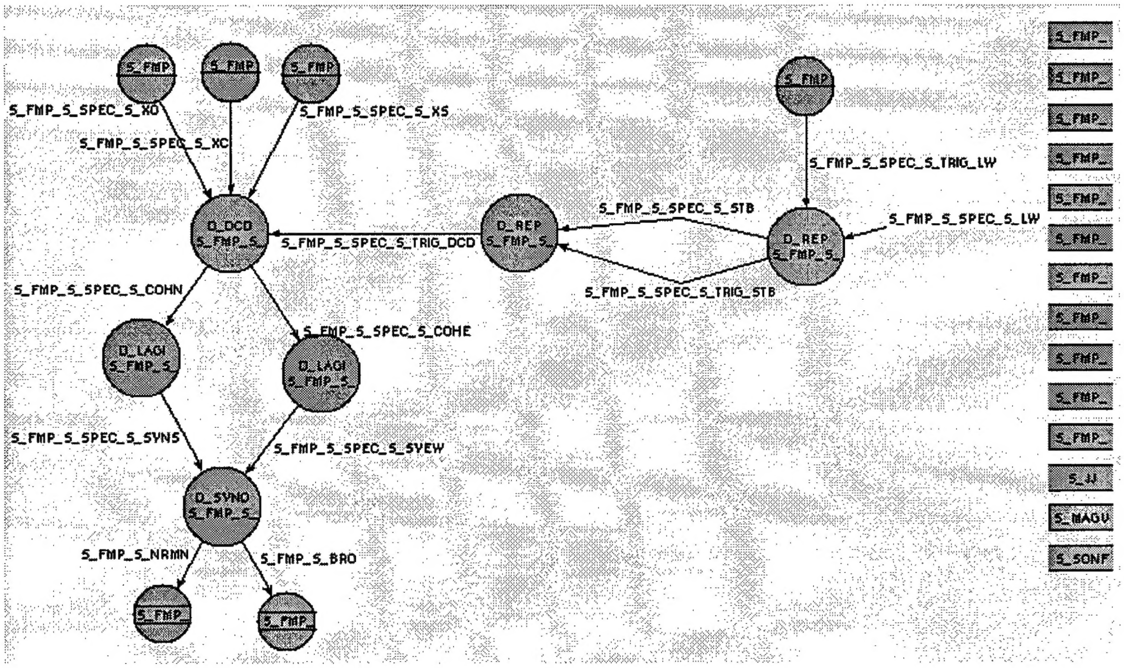


Figure 139. Partition P_FMSPEC_4

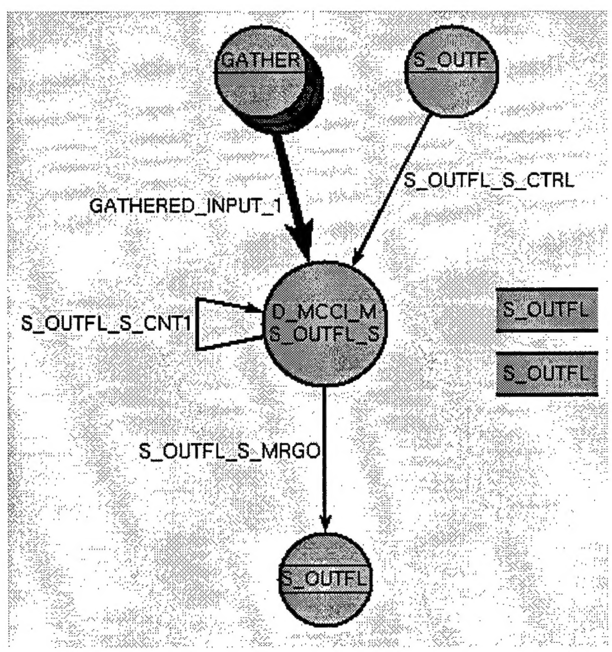


Figure 140. Partition P_OUT_1

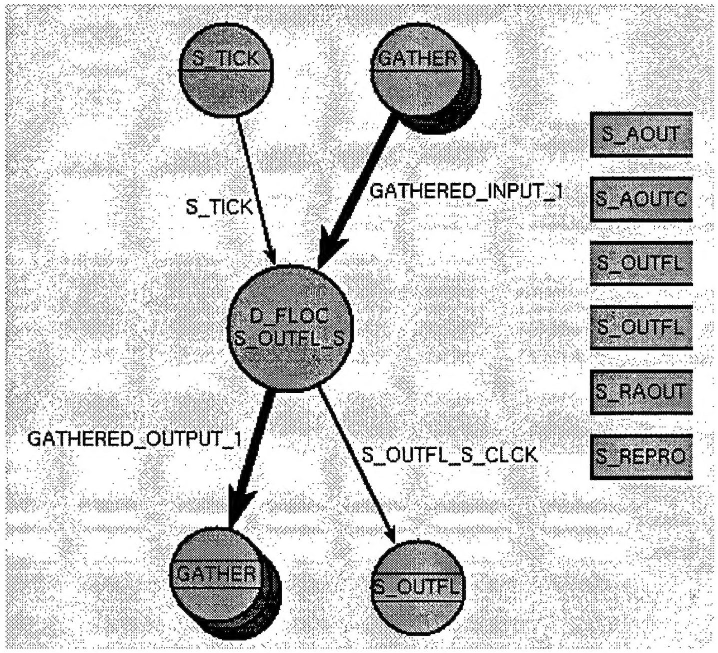


Figure 141. Partition P_OUT_2

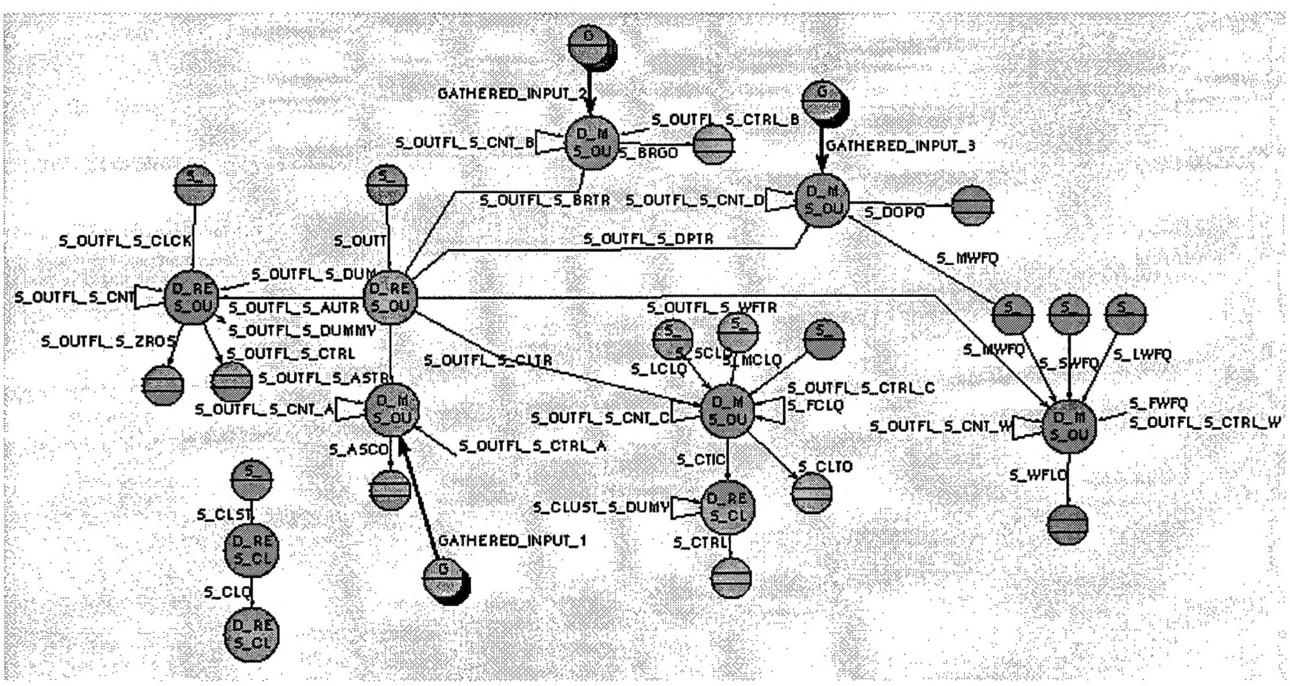


Figure 142. Partition P_OUT_3

Management Communications and Control, Inc. (MCCI)
Contract N68335-98-C-0140

Non-proprietary Abstract - SBIR N98-030 Phase I Final Technical Report

The need for application software portability and reusability has been increased by the COTS revolution. Operating system and math library independence are essential to portability strategies. However, in order to achieve the high throughput required by real-time sensor processing systems, the executable must be optimized for the specific target.

Management Communications and Control, Inc. (MCCI) has developed a methodology and a toolset which provides translation of target independent applications to target specific source code incorporating target optimized libraries. Application portability and reusability is inherent in the methodology. An order of magnitude reduction in application development time has been demonstrated. Life cycle costs should be reduced by at least the same factor. The methodology supports low cost reuse of the AN/UYS-2 code base. This report provides an overview of the methodology and the toolset. Porting of the DICASS sonobuoy signal processing from an AN/UYS-2 implementation to an implementation using the MCCI methodology and toolset is demonstrated.